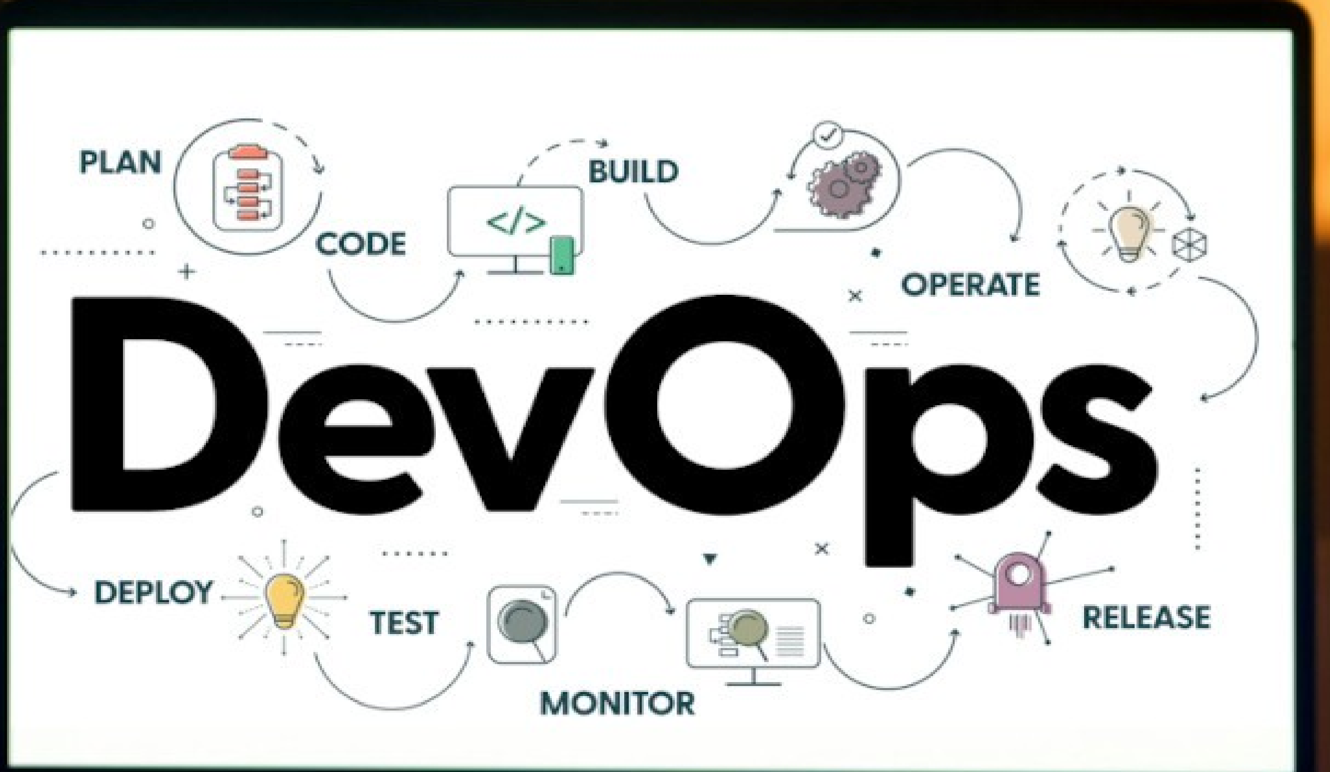
து. நித்யா

எளிய தமிழில்



எளிய தமிழில் DevOps

து.நித்யா

nithyadurai87@gmail.com

மின்னூல் வெளியீடு : கணியம் அறக்கட்டளை kaniyam.com

அட்டைப்படம், மின்னூலாக்கம் : த. சீனிவாசன் tshrinivasan@gmail.com

முதல் பதிப்பு அக்டோபர் 2024

DevOps துறையில் பயன்படுத்தப்படும் ஒருசில முக்கியக் கருவிகள் பற்றி இந்த நூல் எளிமையாக அறிமுகம் செய்கிறது.

தமிழில் கட்டற்ற மென்பொருட்கள் பற்றிய தகவல்களை “கணியம்” மின் மாத இதழ், 2012 முதல் வெளியிட்டு வருகிறது.இதில் வெளியான DevOps பற்றிய கட்டுரைகளை இணைத்து ஒரு முழு புத்தகமாக வெளியிடுவதில் பெரு மகிழ்ச்சி கொள்கிறோம்.

உங்கள் கருத்துகளையும், பிழை திருத்தங்களையும் editor@kaniyam.com க்கு மின்னஞ்சல் அனுப்பலாம்.

https://freetamilebooks.com/learn-devops-in-tamil என்ற முகவரியில் இருந்து இந்த நூலை பதிவிறக்கம் செய்யலாம். உங்கள் கருத்துகளையும் இங்கே பகிரலாம்.

படித்து பயன் பெறவும், பிறருடன் பகிர்ந்து மகிழவும் வேண்டுகிறோம்.

கணியம் இதழை தொடர்ந்து வளர்க்கும் அனைத்து அன்பர்களுக்கும் எமது நன்றிகள்.

த.சீனிவாசன்

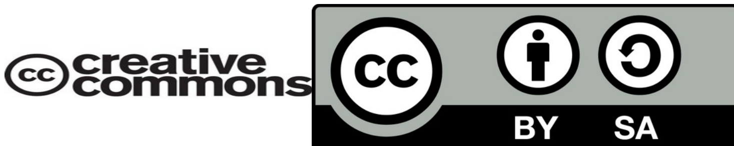
tshrinivasan@gmail.com

ஆசிரியர்
கணியம்

editor@kaniyam.com

பொருளடக்கம்

1. உரிமை



2. ஆசிரியர் உரை

நான் மெஷின் லேர்னிங் துறையில் வேலை பார்க்கும் போது, நான் எழுதிய நிரலை ஒரு அப்ளிகேஷனாக மாற்றி அதனை சர்வரில் ஹோஸ்ட் செய்யும் வாய்ப்பும் எனக்கே கிட்டியது. அப்போது நான் கற்றுக்கொண்ட விஷயங்களே இப்புத்தகத்தை எழுத எனக்கு வித்திட்டது. அதுவரை Devops என்றால் அது ஏதோ ஒரு தனி டெக்னாலஜி என்று தான் நினைத்திருந்தேன். ஆனால் நாம் உருவாக்கிய அப்ளிகேஷனை நடைமுறைக்கு கொண்டு வர உதவும் பல்வேறு சிறுசிறு கருவிகளின் தொகுப்பே DevOps என்பதை பின்னர் தான் புரிந்து கொண்டேன்.

நான் எழுதிய நிரல்களை சேமிக்க உதவும் GITHUB, எந்த சர்வரில் வேண்டுமானாலும் அந்நிரல்களை இயக்க உதவும் DOCKER, ஒவ்வொரு முறை நிரலில் மாற்றம் ஏற்படுத்தும் போதும், டாக்கர் மூலம் அதற்கான அப்ளிகேஷனை உருவாக்கும் வேலையை தானியக்கமாக செய்ய உதவும் JENKINS போன்ற கருவிகளே DevOps-ன் அங்கங்களாக விளங்குகின்றன. மேலும் ஒரே மாதிரியான கட்டமைப்பை பல்வேறு கணினிகளில் அமைக்க உதவும் ANSIBLE, பல்வேறு கணினிகளில் இயங்கிக் கொண்டிருக்கும் jobs-ஐ மேலாண்மை செய்ய உதவும் Airflow, இவைதவிர Kafka, Mongodb போன்ற இன்னபிற DevOps கருவிகள் பற்றியும் இப்புத்தகத்தில் காணலாம்.

து. நித்யா

மிசிசாகா
கனடா



06 அக்டோபர் 2024

மின்னஞ்சல்: nithyadurai87@gmail.com

வலைப்பதிவு: https://nithyashrinivasan.wordpress.com

இலவச மின்னூல்கள் ;
https://freetamilebooks.com/authors/nithyaduraisamy

காணொளிகள் - https://www.youtube.com/@NithyaDuraisamy

3. உதாரணங்கள்

இந்த நூலில் உள்ள நிரல் உதாரணங்கள் யாவும்

http://github.com/nithyadurai87/devops_examples

https://github.com/nithyadurai87/docker_example

https://github.com/nithyadurai87/kafka_airflow_ansible

இங்கே உள்ளன.

4. அறிமுகம்

Development மற்றும் operations இரண்டும் இணைந்து ஒருசேர நடைபெறும் நிகழ்வுகளின் தொடர்ச்சிகளே DevOps என்று அழைக்கப்படுகிறது. வாடிக்கையாளர்கள் கேட்கின்ற விஷயத்தை உருவாக்கித் தருபவருக்கு developer என்று பெயர். இவர் தம்முடைய இடத்தில் (local server) உருவாக்கிய ஒன்றை, வாடிக்கையாளர்களுடைய இடத்தில் (Production server) சிறப்பாக இயங்குமாறு செய்யும் குழுவிற்கு Operations team என்று பெயர். இவ்விரண்டு வேலையையும் ஒருவரே செய்தால் அவரே Devops Engineer என்று அழைக்கப்படுவார். எடுத்துக்காட்டாக உணவகங்களில் நாம் கேட்கின்ற இட்லி, தோசை போன்றவற்றை ஓரிடத்தில் உட்கார்ந்து செய்து கொடுப்பவரை டெவலப்பர் எனலாம். அவர் செய்து கொடுத்ததை நாம் சாப்பிடுவதற்கு ஏற்ற வகையில் ஒரு தட்டில் வைத்து அதனுடன் சட்னி சாம்பார் போன்றவற்றையும் சேர்த்து நம்முடைய டேபிளில் கொண்டு வந்து வைக்கும் சர்வர் போன்றோரை ஆப்பரேஷன்ஸ் டீம் எனலாம். இவ்விரண்டு வேலைகளும் தனித்தனியே நடைபெறாமல் ஒரு சில உணவகங்களில் உள்ள செல்ஃப் சர்வீஸ் முறையை DevOps-க்கு உதாரணமாகக் கூறலாம்.

செல்ஃப் சர்வீஸ் முறையில் டெவலப்பர் தோசையை உருவாக்கியவுடன், அவரே அதனை ஒரு தட்டில் இட்டு அதனோடு சட்னி-சாம்பார் போன்ற துணை உணவுப் பொருட்களையும் சேர்த்து உண்பதற்கு ஏற்ற வகையில் உணவுத்தட்டை தயார் செய்வார். அதேபோல் டெவலப்பர் அப்ளிகேஷனை உருவாக்கியவுடன் அதற்குத் தேவையான துணை packages,os என எல்லாவற்றையும் சேர்த்து ஒரு container-ல் இட்டு, docker என்ற ஒன்றைத் தயார் செய்வர். உணவையும், உணவுத்தட்டையும் தயார் செய்தபின், அதற்கான

டோக்கன் எண்ணை அறிவிப்புப் பலகையில் வெளிப்படுத்தி 'சாப்பிடுவதற்குத் தயாரான நிலையில் தோசை உள்ளது' என்பதை பயனருக்குத் தெரிவிப்பார். அதேபோல் அப்ளிகேசனையும், அதற்கான docker-ஐயும் உருவாக்கியபின், Jenkins மூலம் வேண்டிய இடத்தில் (dev/test/stg/prod environments) நிறுவி, 'பயன்படுத்துவதற்குத் தயாரான நிலையில் அப்ளிகேஷன் உள்ளது' என்பதை பயனர்களுக்குத் தெரிவிப்பார். ஆகவே development + operations இரண்டும் இணைந்து DevOps என்று அழைக்கப்படுகிறது. இவற்றை தனித்தனியாக செய்யாமல் தானியக்க முறையில் தொடர் நிகழ்வுகளாக அமைப்பதன் முக்கியத்துவத்தை அறிய முதலில் ஒரு மென்பொருள் உருவாக்கத்தில் உள்ள முக்கியப் படிநிலைகளை நன்கு புரிந்து கொள்ள வேண்டும். இவை பின்வருமாறு.

1.முதலில் வாடிக்கையாளருக்கு என்னென்ன தேவை என்பதை அவரிடமிருந்து எழுத்துவடிவில் பெற்றுக் கொண்டு அதனை ஆராய்வது 'Requirements gathering and Analysis' எனப்படும்.

2.அடுத்ததாக எப்படி செய்யப் போகிறோம் எனத் திட்டமிடுவது 'Project Planning' எனப்படும்.

3.பின்னர் நிரல்கள் எழுதி தேவையானவற்றை உருவாக்கிக் கொடுப்பதே 'Development' ஆகும். குழுவாக இணைந்து நிரலாளர்கள் வேலை செய்யும்போது, ஒவ்வொருவரும் அவரவர்களுடைய நிரலைப் பகிர்ந்து கொள்வதற்கு SVN, GIT போன்றவற்றைப் பயன்படுத்துகின்றனர். இத்தகைய நிரல்களை ஒருங்கிணைத்து, அவற்றின் செயல்பாடுகளையெல்லாம் சோதித்து, அதில் திருப்தி ஏற்பட்டவுடன் ஒருங்கிணைந்த நிரலை இறுதியாக GIT-ல் பகிர்வர்.

4. GIT-ல் இருக்கும் நிரலை பயன்படுத்தும் வகையிலான அப்ளிக்கேஷனாக மாற்றும் செயலுக்கு build என்று பெயர். Operations குழுவில் இருக்கும் ஒருவர் GIT-லிருந்து நிரலை எடுத்து build செய்து அதனை test environment-ல் நிறுவிக் கொடுப்பர்.

5. Tester என்பவர் அப்ளிகேஷனை சோதித்து வழுக்களையெல்லாம் issue trackerல் பதிவிடுவர். இவற்றையெல்லாம் திருத்தி, திருத்தப்பட்ட நிரலை மீண்டும் GIT-ல் பதிவேற்றுவர் டெவலப்பர். இந்தப் புதிய நிரலை எடுத்து மீண்டும் அப்ளிகேஷனை உருவாக்கி டெஸ்டிங் டீமிடம் கொடுப்பர் ஆப்பரேஷன்ஸ் டீம். இச்சுழற்சியே மீண்டும் மீண்டும் நிகழ்ந்து டெஸ்டரை திருப்தி படுத்தியவுடன் அடுத்த நிலைக்குச் செல்கிறது அப்ளிகேஷன்.

6. டெஸ்டர் அனுமதி கொடுத்தவுடன், நேரடியாக production-ல் சென்று நிறுவாமல், staging என்ற இடத்தில் build செய்து கொடுப்பர் ஆப்பரேஷன்ஸ் டீம். Production சூழல் மாதிரியே உள்ள இந்தச் சூழலில் மீண்டும் ஒருமுறை சோதனை ஓட்டம் (trial run) நிகழ்த்தப்படுகிறது. UAT test, mock test, functional test, regression test போன்ற அனைத்து வகையான சோதனைகளும் மனிதர்கள் மூலமாகவும் தானியக்க முறையிலும் நிகழ்த்தப்பட்டு பிழைகள் கண்டறியப்படுகின்றன. டெஸ்டிங் போது வராத பிழைகள் கூட இந்த environment-ல் வர வாய்ப்பு உள்ளது. ஏனெனில் இது production-ஐ ஒத்தவாறு அமைக்கப்பட்டுள்ளது. ஆகவே பிழைகளை Issue tracker-ல் பதிவிடுவது, பிழை நீக்கம், திருத்தப்பட்ட நிரல்களின் கிட் பதிவேற்றம், மீண்டும் build செய்து environment-ல் நிறுவுதல், மீண்டும் சோதனை, சோதனை, சோதனை மேல் சோதனை என அனைத்தும் இங்கு நிகழ்த்தப்படுகின்றன.

7. கடைசியாக எல்லாப் பிழைகளும் நீக்கப்பட்டபின், production environment-ல் பயன்பாட்டுக்காக நிறுவப்படுகிறது. இங்கும் சில சோதனைகள் நடத்தி சரிபார்த்த பின்னரே அனைவருக்கும் அறிவிக்கப்படுகிறது.

பண்டைய காலங்களில் 'வாட்டர் ஃபால் மாடல்' என்ற முறை பயன்படுத்தப்பட்ட போது மேற்கண்ட அனைத்து செயல்களும் ஒன்றன்பின் ஒன்றாக திகழும். இதனால் ஏற்படும் கால விரயம் மிக அதிகம். ஆனால் தற்போதோ Agile எனும் அதிவேக முறையில் மேற்கண்ட அனைத்தும் இரு வார கால sprint-க்குள் நடைபெறுகின்றன. இதில் உருவாக்கப்படும் அளவிற்குத்தான் requirements-ம் தரப்படுகின்றன. ஆகவே இக்குறுகிய காலத்தில் டெவலப்மெண்ட், டெஸ்டிங், ஆபரேஷன்ஸ் ஆகிய மூன்று குழுக்கள் ஒன்றிணைந்து அதிவிரைவாக செயல்பட வேண்டியது அவசியமாகிறது. இதில் production deployment அடிக்கடி நடக்காவிட்டாலும் கூட, அடிக்கடி நடக்கின்ற dev, test, staging deployment ஆகியவற்றை தானியக்கம் செய்வதன்மூலம் நேர விரயத்தைக் குறைக்கலாம். இதற்கான automation-தான் DevOps என்று அழைக்கப்படுகிறது. மேற்கண்ட படிநிலைகளில் டெவலப்பர் நிரல் எழுதி முடித்த உடனேயே, தாமாக டெவ், டெஸ்ட் அல்லது ஸ்டேஜ்-ல் deploy- ஆகி சோதனைகளை நடத்தி, சோதனை முடிவுகளை குழுவுக்கு அறிவிக்கும் தானியக்க தொடர் செயல்களையே DevOps என்கிறோம். இதில் ஈடுபடுகின்ற முக்கிய மென்பொருட்கள் பின்வருமாறு:

Version control system - CVS, SVN, Mercurial, GIT

Build systems - Jenkins, shell script

Continuous Integration & Continuous Deployment (CI & CD) - Docker, Kubernetes

Testing - Selenium

Alerting - Prometheus, Email, SMS

இனிவரும் பகுதிகளில் ஒரு அப்ளிகேஷனை எவ்வாறு உருவாக்குவது, அதற்கான நிரல்களை எவ்வாறு GIT-ல் சேமிப்பது, Docker file-ஐ எவ்விதம் எழுதுவது, ஜென்கின்ஸ் மூலம் எப்படி அதனை பில்டு செய்து ஒரு சர்வரில் ஹோஸ்ட் செய்வது போன்ற அனைத்தையும் ஒன்றன் பின் ஒன்றாகக் காணலாம்.

5. Application Development

இங்கு இரண்டு விதமான அப்ளிகேஷனை நாம் உருவாக்கப்போகிறோம் . முதலில் ஒரு எடுத்துக்காட்டுக்காக சிம்பிளான ஒரு அப்பிளிக்கேஷன்.. அடுத்து நிஜத்தில் ஒரு நோக்கத்துக்காக உருவாக்கப்படும் சற்று கடினமான அப்பிளிக்கேஷன்.

5.1 Sample Application

'Hello World' என்பதனை பிரிண்ட் செய்யும் ஒரு சாதாரண புரோகிராம் பின்வருமாறு..

sample.py

```
print ("Hello world")
```

இவ்வார்த்தையை வெறும் திரையில் பிரிண்ட் செய்யாமல், ஏதாவதொரு port-ல் வெளியிடுமாறு செய்ய வேண்டும்.. அப்போதுதான் வேறு ஏதாவது அப்ளிகேஷன் நம்முடைய அப்ளிகேஷனுடன் தொடர்புகொண்டு வேண்டியதை பெற்றுக் கொள்ள முடியும்.. இதுவே API என்று அழைக்கப்படும்.. இதற்கு உதவுவதே Flask API ஆகும். நமது கணினியில் 5500 எனும் போர்ட்டில் இவ்வார்த்தையை வெளியிடுவதற்கான நிரல் பின்வருமாறு..

https://gist.github.com/nithyadurai87/1f68117044c66fc3a5b36159a4bba7fc

sample_api.py

```
import os
from flask import Flask

app = Flask(__name__)
port = int(os.getenv('PORT', 5500))

@app.route('/')
def home():
    return render_template('index.html')

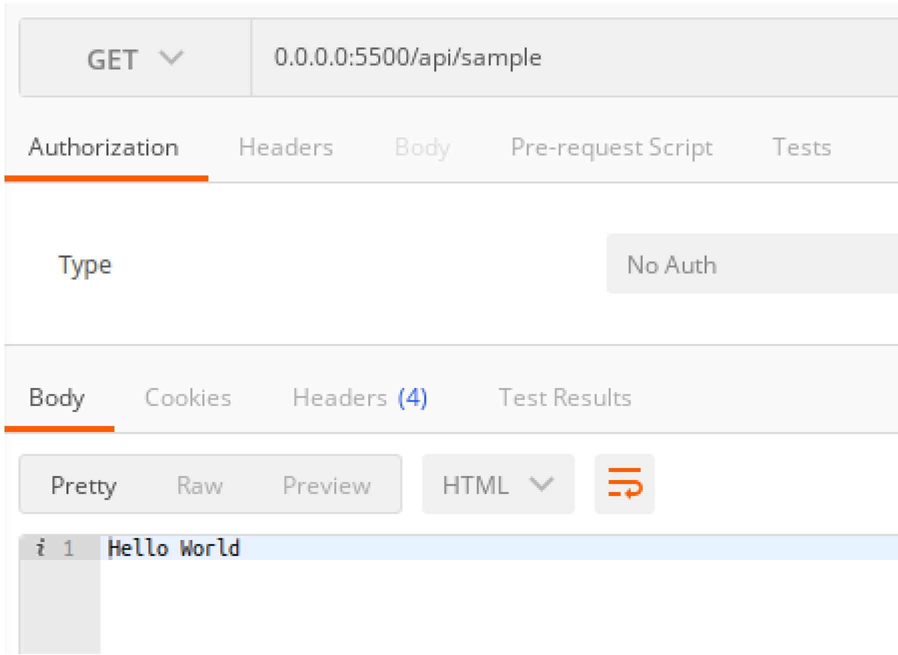
@app.route('/api/sample', methods=['GET'])
def smp():
    return "Hello World"

if __name__ == '__main__':
   app.run(host='0.0.0.0', port=port, debug=True)
```

மேற்கண்ட நிரலை இயக்கினால் அது பின்வருமாறு தனது service-ஐ வழங்கத் தொடங்கும். அதாவது குறிப்பிட்ட போர்ட்டில் சென்று பார்த்தால் இவ்வார்த்தை வெளிப்பட்டுக் கொண்டிருப்பதைக் காணலாம்..

```
shrini@nithya-Lenovo-ideapad:~/nithya/devops$ python3 sample_api.py
 * Serving Flask app "sample_api" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://0.0.0.0:5500/ (Press CTRL+C to quit)
 * Restarting with inotify reloader
 * Debugger is active!
 * Debugger PIN: 335-503-597
127.0.0.1 - - [16/Jun/2020 10:42:07] "GET /api/sample HTTP/1.1" 200 -
```

Postman எனும் கருவி மூலம் இதனை நாம் சரிபார்த்துக் கொள்ளலாம்..



5.2 Real Time Application

எளிய தமிழில் மெஷின் லேர்னிங் என்ற புத்தகத்தில் கொடுத்துள்ள அதே உதாரணத்தை இங்கும் நான் பயன்படுத்தியுள்ளேன்.. இது ஒரு வீட்டின் விற்பனை விலையை நிர்ணயிப்பதற்கான மாடலை உருவாக்குகின்ற ப்ரோக்ராம் ஆகும். இந்த மாடலை உருவாக்குவதற்குத் தேவையான பயிற்சிக்கு பல்வேறு வகையான வீட்டு விலைகளைக் கொண்ட data.csv எனும் கோப்பு உள்ளீடாக கொடுக்கப்பட்டுள்ளது. இதுவே training data ஆகும். இதை வைத்துத்தான் மாடலை உருவாக்கியுள்ளோம்.. அது model.pkl (pickle file) எனும் பெயரில் பைனரி வடிவில் சேமிக்கப்படுகிறது..

https://gist.github.com/nithyadurai87/574937ad8928278057adf43405f213de

model_creation.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import
train_test_split,cross_val_score
#from sklearn.externals import joblib
import joblib
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from math import sqrt
import os

df = pd.read_csv('./data.csv')

i = list(df.columns.values)
i.pop(i.index('SalePrice'))
df0 = df[i+['SalePrice']]
df = df0.select_dtypes(include=['integer','float'])
X = df[list(df.columns)[:-1]]
y = df['SalePrice']
X_train, X_test, y_train, y_test = train_test_split(X, y)

regressor = LinearRegression()
regressor.fit(X_train, y_train)
joblib.dump(regressor, './model.pkl')
```

இப்போது ஒரு சில parameters-ஐ இந்த மாடலுக்குக் கொடுத்தால், அது பெற்றுக்கொண்ட பயிற்சியின் அடிப்படையில் ஒரு விலையைக் கணிக்கும். அதற்கான புரோகிராம் பின்வருமாறு.

https://gist.github.com/nithyadurai87/5db27ddcd1172bf30ffdec42ea633239

prediction.py

```
import os
import json
import pandas as pd
import numpy
#from sklearn.externals import joblib
import joblib

s = pd.read_json('./parameters.json')
p = joblib.load("./model.pkl")
r = p.predict(s)
print (str(r))
```

Json வடிவில் வரும் தரவுகளை எடுத்து, process செய்து, விடையை வெளிப்படுத்தும் நிகழ்வுக்கு serialization மற்றும் de-serialization என்று பெயர். இதற்கு உதவும் வகையில் மாடலை உருவாக்க joblib பயன்பட்டுள்ளது.

அடுத்ததாக மேற்கண்ட புரோகிராம் கணிக்கும் மதிப்பு பிரிண்ட் மூலம் திரையில் வெளிப்படுத்தப்படுகிறது. ஆனால் நிஜத்தில் இம்மதிப்பை வேறு எங்கோ செலுத்த வேண்டி வரும். அது ஒரு டேட்டாபேஸாக இருக்கலாம் அல்லது ஒரு அப்ளிகேஷனாக இருக்கலாம். இவையெல்லாம் வெவ்வேறு இடங்களில் இயங்கிக்கொண்டிருக்கலாம். ஆகவே வெவ்வேறு இடங்களில், பல்வேறு வடிவங்களில் இயங்குகின்ற டேட்டாபேஸ், பல்வேறு மொழிகளில் எழுதப்பட்ட அப்பிளிக்கேஷன் போன்ற எதிலிருந்து வேண்டுமானாலும் இம்மதிப்பை எடுத்துக் கொள்வதற்கு ஏற்ற வகையில் நம்முடைய நிரலை நாம் மாற்ற வேண்டும். அதாவது வேறு எந்த அப்ளிகேஷன் வேண்டுமானாலும் நாம் உருவாக்கிய இந்த அப்ளிகேஷனுடன் பேசி மதிப்பினைப் பெற்றுக்கொள்ளும் செயலுக்கு அப்ளிகேஷன் புரோகிராமிங் இன்டர்ஃபேஸ் (API) என்று பெயர். இதனை வழங்கும் வகையில் நம்முடைய நிரலை

மாற்றுவதற்கு Flask API பயன்படுகிறது. இதைக்கொண்டு உருவாக்கப்பட்ட prediction ப்ரோக்ராம் பின்வருமாறு.

https://gist.github.com/nithyadurai87/457c95dec830c3527f5780f289992133

prediction_api.py

```
import os
import json
import pandas as pd
import numpy
from flask import Flask, render_template, request, jsonify
from pandas.io.json import json_normalize
#from sklearn.externals import joblib
import joblib

app = Flask(__name__)
port = int(os.getenv('PORT', 5600))

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/api/salepricemodel', methods=['POST'])
def salepricemodel():
    if request.method == 'POST':
        try:
            post_data = request.get_json()
            json_data = json.dumps(post_data)
            s = pd.read_json(json_data)
            p = joblib.load("./model.pkl")
```

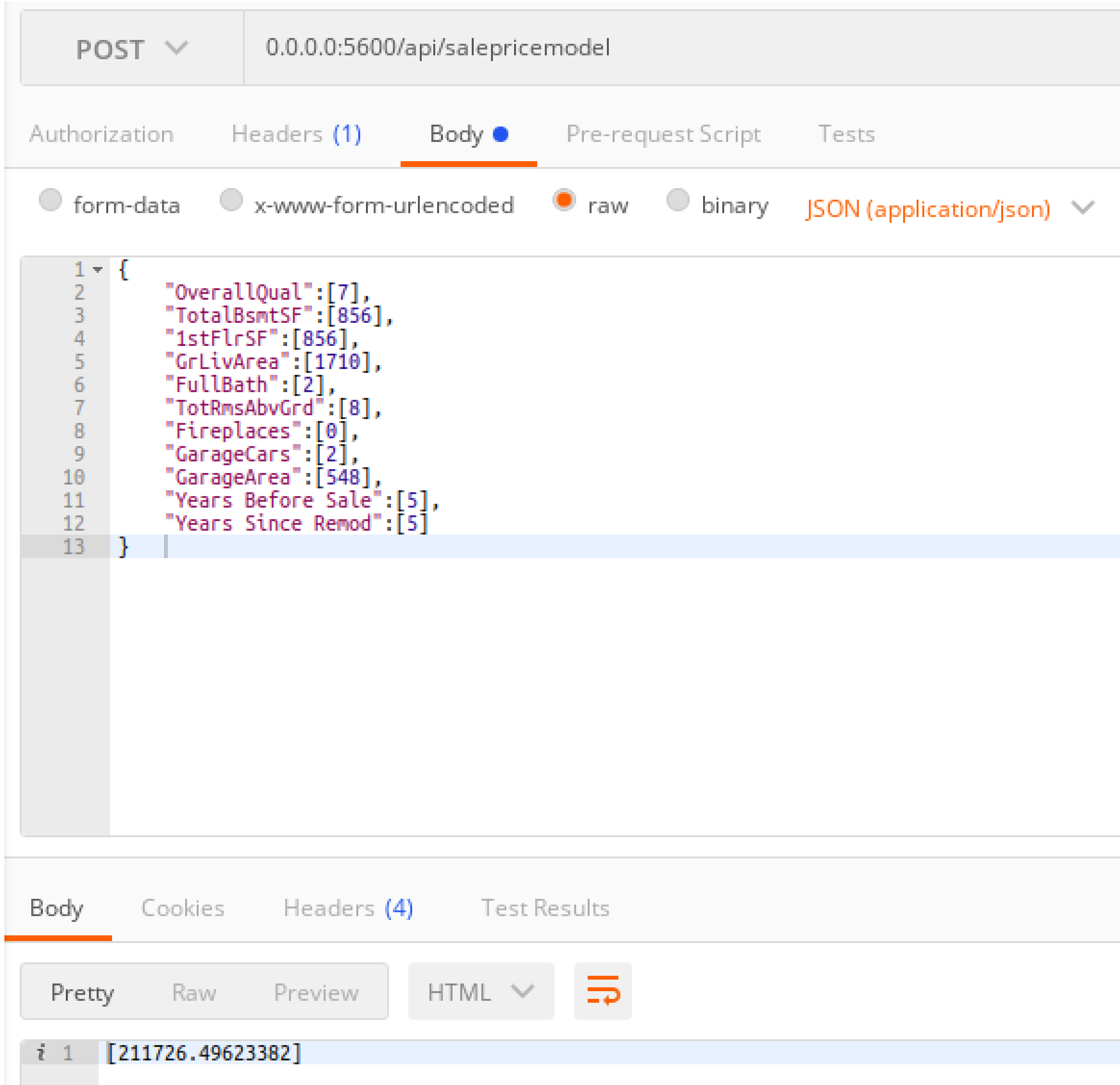
```
            r = p.predict(s)
            return str(r)

        except Exception as e:
            return (e)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=port, debug=True)
```

இந்தப் புரோகிராமை பின்வருமாறு ரன் செய்தால் அது நம்முடைய கணினியில் 5600 எனும் போர்ட் வழியே இதற்கான API-ஐ இயக்கிக் கொண்டிருக்கும். போஸ்ட்மேன் எனும் கருவி மூலம் இந்தப் போர்ட்டில் மாடலுக்கான input json-ஐ அளித்து விடை ஒழுங்காக வருகிறதா என சோதித்துக் கொள்ளலாம். இவை பின்வருமாறு.

```
shrini@nithya-Lenovo-ideapad:~/nithya/devops$ python3 prediction_api.py
 * Serving Flask app "prediction_api" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://0.0.0.0:5600/ (Press CTRL+C to quit)
 * Restarting with inotify reloader
 * Debugger is active!
 * Debugger PIN: 111-784-357
127.0.0.1 - - [19/Jun/2020 12:54:10] "POST /api/salepricemodel HTTP/1.1" 200 -
```



இப்போது மாதிரிக்கு ஒன்று மற்றும் நிஜ அப்ளிகேஷனுக்காக ஒன்று என இரண்டு API நமது கணினியில் தயாராகிவிட்டது. அவை 5500, 5600 எனும் இரண்டு port-ல் மதிப்புகளை வெளிப்படுத்தும் வண்ணம் அமைக்கப்பட்டுள்ளன. அடுத்து இதனை எவ்வாறு GIT-ல் பதிவேற்றம் செய்வது என்று பார்க்கலாம்.

6. GIT

பலரும் இணைந்து ஒரு மென்பொருளை உருவாக்கும்போது, அதன் மூல நிரலில் ஏற்பட்ட மாறுதல்கள், யார் எப்போது மாற்றியது, ஒரே நேரத்தில் யார் யாரெல்லாம் திருத்தியது, எது சமீபத்தியது போன்ற அனைத்தையும் வரலாறு போன்று சேமிக்க உதவும் version control சிஸ்டமே GIT ஆகும். நம்முடைய நிரல்கள் சேமிக்கப்பட்டுள்ள பகுதியில் .git எனும் ஃபோல்டரை உருவாக்கி அதற்குள் இத்தகைய மாற்றங்களை சேமித்துக் கொண்டே வரும். மாற்றங்கள் மட்டுமே இங்கு சேமிக்கப்படுவதால், அதிக இடம் தேவையில்லை. GitHub, GitLab, BitBucket போன்றவை இதுபோன்ற பல்வேறு கிட் களை சேமிக்க உதவும் களஞ்சியம் ஆகும். இது ஒரு மைய சர்வரைக் கொண்டு அனைத்து கிட் களையும் தனக்குள் சேமித்து வைத்துக் கொள்கிறது. எனவே நமது கணினி பழுதடைந்தால்கூட இந்த மைய சர்வரில் சென்று நம்முடைய நிரலைப் பெற்றுக் கொள்ளலாம். தானியக்க முறையில் deployment நிகழ்வதற்கு இதுவே முதல் படி ஆகும்.

6.1 Local server

நமது கணினியில் GIT-ஐ நிறுவுவற்கான மற்றும் நீக்குவதற்கான கட்டளைகள் பின்வருமாறு.

```
sudo apt-get install git
sudo apt-get remove git
```

இப்போது மேற்கண்ட நிரல்கள் அனைத்தும் அமைந்திருக்கும் /nithya/devops எனும் டைரக்டரியில் சென்று git init எனக் கொடுத்தால் அது ஒரு புதிய கிட் repository-ஐ உருவாக்கும்.

```
$ git init
```

```
shrini@nithya-Lenovo-ideapad:~/nithya/devops$ ls
data.csv  Dockerfile  model_creation.py  model.pkl  parameters.json  prediction_
api.py  prediction.py  requirements.txt  sample_api.py  sample.py
shrini@nithya-Lenovo-ideapad:~/nithya/devops$ git init
Initialized empty Git repository in /home/shrini/nithya/devops/.git/
```

இப்போது நம்முடைய டைரக்டரியில் சென்று பார்த்தால் .git எனும் ஃபோல்டர் மறைமுகமாக இடம் பெற்றிருப்பதைக் காணலாம். இதுவே hidden folder ஆகும். இதற்குள் தான் அனைத்து செயல்களும் பதிவு செய்யப்பட்டுக் கொண்டே வரும். நம்முடைய டைரக்டரி ஒரு git repository-ஆக உருமாற்றம் அடைந்ததை இது உறுதிப்படுத்தும்.

```
shrini@nithya-Lenovo-ideapad:~/nithya/devops$ ls -la
total 260
drwxrwxr-x  3 shrini shrini   4096 Jun 19 11:12 .
drwxr-xr-x 11 shrini shrini   4096 Jun 11 09:58 ..
-rw-rw-r--  1 shrini shrini 215095 Jun 11 10:04 data.csv
-rw-rw-r--  1 shrini shrini    181 Jun  9 14:25 Dockerfile
drwxrwxr-x  7 shrini shrini   4096 Jun 19 11:12 .git
-rw-rw-r--  1 shrini shrini    702 Jun 11 10:26 model_creation.py
-rw-rw-r--  1 shrini shrini    697 Jun 11 10:28 model.pkl
-rw-rw-r--  1 shrini shrini    248 Jun 11 10:04 parameters.json
-rw-rw-r--  1 shrini shrini    871 Jun 11 19:33 prediction_api.py
-rw-rw-r--  1 shrini shrini    221 Jun 11 19:34 prediction.py
-rw-rw-r--  1 shrini shrini     47 Jun  9 14:24 requirements.txt
-rw-rw-r--  1 shrini shrini    364 Jun 16 10:41 sample_api.py
-rw-rw-r--  1 shrini shrini     23 Jun 16 10:34 sample.py
shrini@nithya-Lenovo-ideapad:~/nithya/devops$ cd .git
shrini@nithya-Lenovo-ideapad:~/nithya/devops/.git$ ls
branches  config  description  HEAD  hooks  info  objects  refs
```

GIT மூன்று படிநிலைகளில் கோப்புகளை அணுகும். முதலாவது நம்முடைய தற்போதைய டைரக்டரி, இரண்டாவது staging பகுதி, மூன்றாவது git-க்கான மைய சர்வர். git status எனும் கட்டளை எந்தப் படி நிலையில் நமது கோப்புகள் உள்ளன என்பதை வெளிப்படுத்தும்.

```
$ git status
```

```
shrini@nithya-Lenovo-ideapad:~/nithya/devops/.git$ cd ..
shrini@nithya-Lenovo-ideapad:~/nithya/devops$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Dockerfile
        data.csv
        model.pkl
        model_creation.py
        parameters.json
        prediction.py
        prediction_api.py
        requirements.txt
        sample.py
        sample_api.py

nothing added to commit but untracked files present (use "git add" to track)
```

இப்போதுதான் git-ஐ உருவாக்கியுள்ளோம். இன்னும் எதையும் git-க்கு அறிமுகம் செய்யவில்லை. அனைத்து கோப்புகளும் தற்போதைய டைரக்டரியில் தான் உள்ளன. ஆகவே 'எதைப் பற்றிய விவரமும் எனக்குத் தெரியவில்லை' என்பது போல் அனைத்தையும் சிகப்பு நிறத்தில் வெளிப்படுத்தியுள்ளது. ஆகவே git add எனக் கொடுத்து அனைத்தையும் git-க்கு அறிமுகம் செய்யப் போகிறோம்.

```
$ git add * .
```

இதன்பின் கோப்புகள் அனைத்தும் ஸ்டேஜிங் பகுதியை சென்றடைந்து விடும். ஸ்டார் டாட் என்பது தற்போதைய டைரக்டரியில் உள்ள அனைத்துக் கோப்புகளையும் நகர்த்த வேண்டும் என்பதைக் குறிக்கும். ஏதேனும் ஒரு கோப்பை மட்டும் நகர்த்த விரும்பினால், dot-க்கு முன்னர் அக்கோப்பின் பெயரை மட்டும் அளித்தால் போதும்.

```
shrini@nithya-Lenovo-ideapad:~/nithya/devops$ git add * .
shrini@nithya-Lenovo-ideapad:~/nithya/devops$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   Dockerfile
        new file:   data.csv
        new file:   model.pkl
        new file:   model_creation.py
        new file:   parameters.json
        new file:   prediction.py
        new file:   prediction_api.py
        new file:   requirements.txt
        new file:   sample.py
        new file:   sample_api.py
```

இந்நிலையில் கோப்புகளைப் பற்றிய அறிமுகம் தனக்குக் கிடைத்து விட்டதால் அனைத்தையும் பச்சை நிறத்தில் வெளிப்படுத்தியுள்ளதைக் காணலாம்.

இங்கு நாம் உருவாக்கிய நிரல்களோடு சேர்த்து data.csv , model.pkl என அனைத்தையும் staging இடத்தில் ஏற்றியுள்ளோம். ஆனால் உண்மையில் இவ்வாறு செய்யக்கூடாது. வெறும் நிரல்களை மட்டும் தான் இங்கு சேமிக்க வேண்டும். நிரல்களுக்குள் தர வேண்டிய உள்ளீட்டுத் தரவுகளை வேறு எங்காவது சேமித்து அதற்கான API மூலம் அதனை நிரலுக்குள் கொண்டு வர வேண்டும். அதேபோல் நாம் உருவாக்கிய மாடலையும் எங்கு சேமிக்க விரும்புகிறோமோ அதற்கான API எழுதி அங்கு கொண்டுபோய் சேமிக்க வேண்டும். இதுவே சரியான முறையாகும். இங்கு நாம் செய்து பார்ப்பதற்கு வசதியாக இருக்க வேண்டுமென்று அனைத்தையும் நான் பதிவேற்றி உள்ளேன்.

அடுத்து வெறும் அறிமுகம் கிடைத்தால் போதுமா! அதனைத் தன்னுள் ஒருவராக இணைத்துக்கொள்ள வேண்டாமா! இதற்காக கமிட் பயன்படுகிறது.

```
$ git commit -m "first commit"
```

இங்கு -m என்பது கமெண்ட் அளிக்கப் பயன்படும்.

```
shrini@nithya-Lenovo-ideapad:~/nithya/devops$ git commit -m "first commit"
[master (root-commit) ea5b3c7] first commit
 10 files changed, 1577 insertions(+)
 create mode 100644 Dockerfile
 create mode 100644 data.csv
 create mode 100644 model.pkl
 create mode 100644 model_creation.py
 create mode 100644 parameters.json
 create mode 100644 prediction.py
 create mode 100644 prediction_api.py
 create mode 100644 requirements.txt
 create mode 100644 sample.py
 create mode 100644 sample_api.py
shrini@nithya-Lenovo-ideapad:~/nithya/devops$ git status
On branch master
nothing to commit, working tree clean
```

இதைத்தொடர்ந்து ஸ்டேஜிங் நிலையில் இருப்பவை கடைசி நிலையான சர்வரைச் சென்று அடைந்து விடும்.

மேற்கண்ட அனைத்தும் நம்முடைய லோக்கல் சர்வரில் தான் நடந்துள்ளன. வேறு யாராவது இவற்றைப் பயன்படுத்த விரும்பினாலோ அல்லது நாமே வேறு சர்வரில் சென்று இதே அமைப்பை நிறுவ விரும்பினாலோ, அதற்கு உதவும் வகையில் நம்முடைய நிரல்கள் அனைத்தையும் ஒரு மைய சர்வரில் வைக்கவேண்டும். GitHub என்ற ஒரு மைய சர்வரில் நம்முடைய நிரல்களைப் பதிவேற்றம் செய்வது பற்றி இப்போது பார்க்கலாம்.

6.2 Centralized Repository

https://github.com/ எனும் முகவரியில் சென்று Repositories -> New என சொடுக்கினால் பின்வருமாறு ஒரு திரை வெளிப்படும். இங்கு devops_examples எனும் பெயரில் ஒரு repository-ஐ உருவாக்கிக் கொள்ளவும். இதுவே நம்முடைய நிரல்களை சேமிக்க உதவும் களஞ்சியமாகச் செயல்படும்.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner **Repository name** *

nithyadurai87 / devops_examples

Great repository names are short and memorable. Need inspiration? How about **effective-carnival**?

Description (optional)

  **Public**
Anyone on the the internet can see this repository. You choose who can commit.

  **Private**
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

 **Initialize this repository with a README**
This will let you immediately clone the repository to your computer.

Add .gitignore: **None** | Add a license: **None** 

Create repository

அதன் முகவரி பின்வருமாறு வெளிப்படும்.

Quick setup — if you've done this kind of thing before

or HTTPS SSH https://github.com/nithyadurai87/devops_examples.git

Get started by creating a new file or uploading an existing file. We recommend every repository include a R

...or create a new repository on the command line

```
echo "# devops_examples" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/nithyadurai87/devops_examples.git
git push -u origin master
```

இந்த முகவரியுடன் நம்முடைய லோக்கல் சர்வரை இணைக்க git remote add எனும் கட்டளை பயன்படுகிறது. இது 2 arguments-ஐப் பெற்று இயங்குகிறது. முதலில் உள்ள origin என்பது இணைக்கப் போகும் சர்வருக்கு நாம் வழங்குகின்ற பெயராகும். வேறு எதை வேண்டுமானாலும் பெயராக நாம் வழங்கலாம். அடுத்து உள்ளது சர்வருக்கான url ஆகும். இந்த இணைப்பு நடந்தவுடன் git push எனக் கொடுத்து அனைத்தையும் நாம் ரிமோட் சர்வருக்கு அனுப்பி விடலாம். இவை பின்வருமாறு.

```
$ git remote add origin
https://github.com/nithyadurai87/devops_examples.git

$ git push origin master
```

```
shrini@nithya-Lenovo-ideapad:~/nithya/devops$ git remote add origin https://github.com/nithyadurai87/devops_examples.git
shrini@nithya-Lenovo-ideapad:~/nithya/devops$ git push origin master
Username for 'https://github.com': nithyadurai87
Password for 'https://nithyadurai87@github.com':
Enumerating objects: 12, done.
Counting objects: 100% (12/12), done.
Delta compression using up to 4 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (12/12), 38.45 KiB | 579.00 KiB/s, done.
Total 12 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/nithyadurai87/devops_examples.git
 * [new branch]      master -> master
```

இங்கு devops எனும் ஒரே ஃபோல்டருக்குள் இரண்டு வெவ்வேறு போர்டில் மதிப்புகளை வெளிப்படுத்தக்கூடிய இரண்டு API-களுக்கான ப்ரோக்ராம்கள் இடம்பெற்றுள்ளன. ஆனால் இதற்கு அடுத்த படியில் ஒவ்வொரு API-க்கும் தனித்தனி Dockerfile- ஐ எழுதி ரன் செய்யப்போகிறோம். ஆகவே இதற்கு வசதியாக இவ்விரண்டையும் பிரித்து தனித்தனியே real_time, sample எனும் இரண்டு ஃபோல்டரில் சேமிக்கிறோம்.

```
shrini@nithya-Lenovo-ideapad:~/nithya/devops$ ls -R
.:
real_time  sample

./real_time:
data.csv  Dockerfile  model_creation.py  model.pkl  parameters.json
prediction_api.py  prediction.py  requirements.txt

./sample:
Dockerfile  sample_api.py  sample.py
```

இதுபோன்று சில மாறுதல்கள் செய்து git status எனக்கொடுத்தாலும் அது பின்வருமாறு வெளிப்படுத்துவதைக் காணலாம்.

```
shrini@nithya-Lenovo-ideapad:~/nithya/devops$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    Dockerfile
        deleted:    data.csv
        deleted:    model.pkl
        deleted:    model_creation.py
        deleted:    parameters.json
        deleted:    prediction.py
        deleted:    prediction_api.py
        deleted:    requirements.txt
        deleted:    sample.py
        deleted:    sample_api.py

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        real_time/
        sample/
```

எனவே மீண்டும் git add, git commit, git push என அனைத்தையும் கொடுத்து எப்போதும் நம்முடைய தற்போதைய directory-ம் ரிமோட்டில் உள்ள directory-ம் ஒன்றுபோல் இருக்குமாறு பார்த்துக் கொள்ள வேண்டும்.

6.3 More Git Commands

அடுத்ததாக கிட்டில் ஏற்கனவே பதிவேற்றம் செய்யப்பட்டுள்ள நிரல்களை பதிவிறக்கம் செய்து பயன்படுத்த கீழ்க்கண்ட கட்டளை பயன்படும்.

```
$ git clone https://github.com/nithyadurai87/docker_example.git
```

இதில் டாக்கர் சம்பந்தப்பட்ட நிரல்கள் சேமிக்கப்பட்டுள்ளன. இனிவரும் பகுதிகளில் இந்நிரல்களுக்கான விளக்கங்களைக் காணலாம். அதற்கு முன்னர் இதைவைத்து இன்னும் சில git கட்டளைகளைப் பற்றித் தெரிந்து கொள்ளலாம்.

பதிவிறக்கம் செய்த உடன் நிரல்களில் ஏதேனும் மாற்றம் செய்ய விரும்பினால் இதில் நேரடியாக மாற்றக்கூடாது. ஏனெனில் இது மாஸ்டர் ஆகும். நமக்கென ஒரு கிளையை உருவாக்கி அதற்குள்தான் நமது வேலைகளைத் தொடங்க வேண்டும்.

கீழ்க்கண்ட கட்டளை volume என்ற பெயரில் கிளையை உருவாக்கும். அதற்கு அடுத்த கட்டளை நாம் எந்தக் கிளையில் உள்ளோம் என்பதை வெளிக்காட்டும். இங்கு நட்சத்திரக் குறியைத் தொடர்ந்து master என்பதை வெளிப்படுத்தியுள்ளது. அதாவது இன்னும் மாஸ்டரில் தான் உள்ளோம். ஆகவே கிட் checkout எனக் கொடுத்து நாம் உருவாக்கிய புதிய branch-க்குச் செல்லலாம்.

```
$ git branch volume
$ git branch
$ git checkout volume
```

```
shrinivasan@shrinivasan-Lenovo-Z50-70:~/nithya/app_stack$ git branch volume
shrinivasan@shrinivasan-Lenovo-Z50-70:~/nithya/app_stack$ git branch
* master
  volume
shrinivasan@shrinivasan-Lenovo-Z50-70:~/nithya/app_stack$ git checkout volume
Switched to branch 'volume'
```

இவ்வாறு தனித்தனியாக அல்லாமல் ஒரே கட்டளையில் branch-ஐ உருவாக்கி உள்நுழைய,

```
$ git checkout -b volume
```

எனவும் கொடுக்கலாம்.

இப்போது app.py, docker-compose.yml ஆகிய கோப்பிற்குள் கமெண்ட்-ஆக உள்ள வரிகளில் கமெண்ட்டை நீக்கிவிட்டு சேமித்துக் கொள்வோம். இதைத் தொடர்ந்து git diff எனக் கொடுத்தால் நிரல்களில் நிகழ்ந்துள்ள மாற்றங்கள் பின்வருமாறு வெளிப்படும்.

```
$ git diff
```

```
shrinivasan@shrinivasan-Lenovo-Z50-70:~/nithya/app_stack$ git diff
diff --git a/app.py b/app.py
index ec1b225..09233ee 100644
--- a/app.py
+++ b/app.py
@@ -12,8 +12,8 @@ def index():
     if request.method == 'POST':
         data = request.get_json()
         collection.insert_one(data).inserted_id
-        #with open('./tmpfiles/data.log','a') as f:
-            #f.write(str(data)+'/n')
+        with open('./tmpfiles/data.log','a') as f:
+            f.write(str(data)+'/n')
         return ('', 204)

     if request.method == 'GET':
diff --git a/docker-compose.yml b/docker-compose.yml
index aad295a..f4c0709 100644
--- a/docker-compose.yml
+++ b/docker-compose.yml
@@ -8,8 +8,8 @@ services:
        - FLASK_ENV=development
     ports:
       - 5000:5000
-    #volumes:
-      #- /home/nithya/backup:/app/tmpfiles
```

இத்தகைய மாற்றங்களை கமிட் செய்வதற்கு முன்னர் git stash எனக் கொடுத்தால் மாற்றங்கள் அனைத்தும் நீங்கி கோப்புகள் பழைய நிலையைச் சென்றடையும்.

```
$ git stash
```

```
shrinivasan@shrinivasan-Lenovo-Z50-70:~/nithya/app_stack$ git stash
Saved working directory and index state WIP on volume: 1c6f119 first
shrinivasan@shrinivasan-Lenovo-Z50-70:~/nithya/app_stack$ git diff
shrinivasan@shrinivasan-Lenovo-Z50-70:~/nithya/app_stack$ git log
commit 1c6f119e6aa674c303d7d60967c26c8e7ad5f827 (HEAD -> volume, origin/master, master)
Author: shrinivasan <shrinivasan@netcalyx.co>
Date:   Thu Oct 8 18:00:00 2020 +0530

    first
```

```
$ git log
```

git log என்பது இதனை பதிவேற்றம் செய்தபோது பயன்படுத்திய பெயர், மின்னஞ்சல் முகவரி போன்ற சில அடிப்படைத் தகவல்களை வெளிப்படுத்தும். இவற்றை மாற்ற விரும்பினால் git config பின்வருமாறு பயன்படும்.

```
$ git config user.name "Nithya Duraisamy"
$ git config user.email "nithyadurai87@gmail.com"
```

```
o-Z50-70:~/nithya/app_stack$ git config user.name "Nithya Duraisamy"
o-Z50-70:~/nithya/app_stack$ git config user.email "nithyadurai87@gmail.com"
```

இப்போது நாம் செய்த மாற்றங்களை நமது branch-க்குள் சேமிக்க பின்வரும் கட்டளைகள் பயன்படும். மேலே நாம் மாற்றிய பயனரின் பெயர் மற்றும் மின்னஞ்சல் முகவரியின் கீழ் இவை சேமிக்கப்படும்.

```
$ git add *
$ git commit -m 'second'
$ git push origin volume
```

```
shrinivasan@shrinivasan-Lenovo-Z50-70:~/nithya/app_stack$ git add * .
shrinivasan@shrinivasan-Lenovo-Z50-70:~/nithya/app_stack$ git commit -m 'second'
[volume a54a205] second
 2 files changed, 4 insertions(+), 4 deletions(-)
shrinivasan@shrinivasan-Lenovo-Z50-70:~/nithya/app_stack$ git push origin volume
Username for 'https://github.com': nithyadurai87
Password for 'https://nithyadurai87@github.com':
Counting objects: 4, done.
```

```
$ git diff master
```

இது மாஸ்டருக்கும் பிராஞ்சுக்கும் உள்ள வேறுபாட்டை வெளிப்படுத்தும்..

ஆகவே மாஸ்டராக உள்நுழைந்து நம்முடைய branch-ஐ இணைத்துக் கொண்டால், நாம் செய்த மாற்றங்கள் மாஸ்டரிலும் இணைந்து விடும்.

```
$ git checkout master
$ git merge volume
```

```
shrinivasan@shrinivasan-Lenovo-Z50-70:~/nithya/app_stack$ git checkout master
Switched to branch 'master'
shrinivasan@shrinivasan-Lenovo-Z50-70:~/nithya/app_stack$ git merge volume
Updating 1c6f119..a54a205
Fast-forward
 app.py             | 4 ++--
 docker-compose.yml | 4 ++--
 2 files changed, 4 insertions(+), 4 deletions(-)
```

இப்பகுதியில் அதிக முக்கியத்துவம் வாய்ந்த கிட் கட்டளைகளைப் பற்றி மட்டும் நாம் பார்த்துள்ளோம். இதுவரை ஒரு அப்ளிகேஷனை உருவாக்குவது , அவற்றின் நிரல்களை git- சேமிப்பது போன்றவற்றைப் பற்றியெல்லாம் பார்த்தோம். ஆனால் அந்த அப்ளிகேஷனைவெற்றிகரமாக ஒரு புதிய இடத்தில் நிறுவுவதற்கு

இது மட்டும் போதாது. புதிய சர்வரில் நிலவும் பல்வேறு config அமைவுகளால் அப்ளிகேஷனின் செயல்பாடு பாதிக்கக்கூடும். அதனை சரி செய்வதற்காக வந்ததே docker ஆகும். இதைப் பற்றி அடுத்த பகுதியில் காணலாம்..

7. Docker

Develop, Ship & Run anywhere என்பதே docker-ன் தத்துவம் ஆகும். ஓரிடத்தில் உருவாக்கப்படும் அப்ளிகேஷனை, இடம் மாற்றி, எங்கு வேண்டுமானாலும் நிறுவி தங்கு தடையின்றி இயங்க வைக்குமாறு செய்ய docker உதவுகிறது. Cloud சிஸ்டம் தனது சேவைகளை மூன்று விதங்களில் வழங்குகிறது. அவை PaaS (P -Platfrom), SaaS (S -Software), IaaS (I -Infrastructure) என்று அழைக்கப்படுகின்றன. இதில் Platform as a service என்பதை வழங்குவதற்குத் தேவையான கருவிகளை உள்ளடக்கியதே docker ஆகும்.

முன்னரெல்லாம் நம்முடைய லோக்கல் சர்வரில் நம்முடைய configurations-ல் உருவாக்கப்பட்ட ஒரு அப்ளிகேஷனை, வேறொரு புதிய சர்வருக்கு அனுப்பும்போது அங்கு நிலவும் configurations-ஆல் அதனால் சரிவர இயங்கி பயனளிக்க முடியாது. ஆகவே இப்பிரச்சினையைத் தீர்க்கும் iபொருட்டு அப்ளிக்கேஷனுக்குத் தேவையான மென்பொருட்கள், லைப்ரரி, config கோப்புகள் மற்றும் OS-ஐக் கூட மெய்நிகராக்கம் (virtualization) செய்து அனைத்தையும் ஒன்றாக மூட்டை கட்டி ஒரு தொகுப்பாக அமைக்கிறது. இதற்கு container என்று பெயர். இதற்குள் இருப்பவற்றை இயக்கி நமது அப்ளிகேஷனின் இயங்கக்கூடிய பதிப்பினை (executable version) உருவாக்கம் செய்வதே image என்று அழைக்கப்படுகிறது. இந்த இமேஜை உருவாக்கத் தேவையான விதிகளைக் கொண்டுள்ள கோப்பு Dockerfile என்று அழைக்கப்படுகிறது.

நமது கணினியில் docker-ஐ நிறுவுவற்கான மற்றும் நீக்குவதற்கான கட்டளைகள் பின்வருமாறு.

```
$ sudo apt-get install docker
$ sudo apt-get remove docker
```

docker-ஐ நிறுவிய பின்னர் நமது கணினியில் தற்போது இயங்கிக்கொண்டிருக்கும் கன்டெய்னர்களைக் காண docker ps எனும் கட்டளை பயன்படும். -a என சேர்த்துக் கொடுத்தால் இயக்கத்தில் இல்லாத கன்டெய்னர்களையும் காட்டும். docker images எனும் கட்டளை இதுவரை உருவாக்கியுள்ள இமேஜ் அனைத்தையும் பட்டியலிடும்.

```
shrini@nithya-Lenovo-ideapad:~/nithya/devops$ docker ps
Got permission denied while trying to connect to the Docker daemon socket at uni
x:///var/run/docker.sock: Get http://%2Fvar%2Frun%2Fdocker.sock/v1.40/containers
/json: dial unix /var/run/docker.sock: connect: permission denied
shrini@nithya-Lenovo-ideapad:~/nithya/devops$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
```

```
shrini@nithya-Lenovo-ideapad:~/nithya/devops$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
prediction          6                   5a49b3a5fd5f        8 days ago          722MB
prediction          v1                  1c1882f0dc1a        9 days ago          722MB
ubuntu              20.04               1d622ef86b13        8 weeks ago         73.9MB
```

இப்போது நான் இதுவரை உருவாக்கியுள்ள இமேஜ் அனைத்தையும் அழித்துவிட்டு புதிதாக ஒரு இமேஜை உருவாக்கப் போகிறேன். ஆகவே

```
$ docker system prune -a
```

எனும் கட்டளை மூலம் இயக்கத்தில் இல்லாத அனைத்து கன்டெய்னர்கள், இமேஜ், நெட்வொர்க் போன்ற அனைத்தையும் நீக்கியுள்ளேன்.

```
shrini@nithya-Lenovo-ideapad:~/nithya/devops$ sudo docker system prune -a
WARNING! This will remove:
  - all stopped containers
  - all networks not used by at least one container
  - all images without at least one container associated to them
  - all build cache

Are you sure you want to continue? [y/N] y
Deleted Containers:
2c00669ae0493b3365cc941283bed3115683d348333866aec40ae1e9428c815a

Deleted Images:
untagged: ubuntu:20.04
untagged: ubuntu@sha256:8bce67040cd0ae39e0beb55bcb976a824d9966d2ac8d2e4bf6
```

இப்போது இமேஜ் எதுவும் வெளிப்படவில்லை.

```
shrini@nithya-Lenovo-ideapad:~/nithya/devops$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
shrini@nithya-Lenovo-ideapad:~/nithya/devops$ ls
real_time  sample
```

அடுத்ததாக இமேஜை உருவாக்குவதற்குத் தேவையான விதிகளை உள்ளடக்கிய டாக்கர் கோப்பினை உருவாக்க வேண்டும்.

7.1 Dockerfile

நாம் விரும்பும் வேலைகளை கணினியை செய்ய வைக்க ஒவ்வொரு கட்டளைகளாகக் கொடுத்து கணினிக்கு உத்தரவு பிறப்பிப்போம். இவ்வாறு பல கட்டளைகளை தனித்தனியே தட்டச்சு செய்வதை விட, அவை அனைத்தையும் ஒரே file-ல் எழுதி ஒரே கட்டளை மூலம் அந்தக் கோப்பில் உள்ள எல்லா commands-ஐயும் ஒன்றன்பின் ஒன்றாக இயங்க வைக்க முடியும். இதுவே லினுக்ஸில் shell scripting என்றும், விண்டோஸில் Batch file என்றும் அழைக்கப்படுகிறது. இதே வேலையை ஒரு container-க்குள் செய்து இமேஜை உருவாக்க உதவுவதே Dockerfile ஆகும்.

இங்கு real_time, sample எனும் இரண்டு ஃபோல்டருக்குள் இரண்டு வெவ்வேறு அப்ளிகேஷன்கள் சேமிக்கப்பட்டுள்ளன. ஆகவே ஒவ்வொரு ஃபோல்டருக்குள்ளும் சென்று அந்தந்த அப்ளிகேஷனை இயக்குவதற்குத் தேவையான விதிகளை உள்ளடக்கிய டாக்கர் கோப்பினை உருவாக்க வேண்டும்.

முதலில் sample எனும் ஃபோல்டருக்குள் சென்று பின்வரும் கட்டளைகளை உள்ளடக்கிய ஒரு கோப்பினை உருவாக்கி அதனை Dockerfile எனும் பெயரில் சேமிக்கவும்.

https://gist.github.com/nithyadurai87/dde5421028333dff0ef52f5bffefb439

Dockerfile

```
FROM ubuntu:20.04
COPY . /app
WORKDIR /app
RUN apt-get update
RUN apt-get install python3-pip -y
RUN pip3 install flask
ENTRYPOINT ["python3"]
CMD ["sample_api.py"]
```

இக்கோப்பிற்குள் கொடுக்கப்பட்டுள்ள கட்டளைகளின் வரிசைப்படி பார்த்தால், முதலில் உபுண்டு 20.04-ன் இமேஜை பிரதியெடுத்து டாக்கருக்குள் உருவாக்கும். பின் தற்போதைய sample டைரக்டரியில் உள்ளதை உபுண்டுவின் /app எனும் டைரக்டரிக்குள் செலுத்தும். பின் அதனை வொர்கிங் டைரக்டரியாக மாற்றி உபுண்டுவை ஒருமுறை அப்டேட் செய்யும். பின் நமது அப்ளிகேஷனுக்குத் தேவையான நிரலாக்க மொழியை நிறுவும். இதன் தொடர்ச்சியாக இன்ஸ்டால்

செய்யப்படவேண்டிய லைப்ரரி அனைத்தையும் ஒன்றன்பின் ஒன்றாக நிறுவுவதற்கான கட்டளைகளை இங்கேயே கொடுத்தாலும் கொடுக்கலாம் அல்லது requirements.txt எனும் பெயரில் தனியாக ஒரு கோப்பினை உருவாக்கி அதிலிருந்து ஒவ்வொன்றாக எடுத்து நிறுவும் படியும் சொல்லலாம். இங்கு flask எனும் ஒரே ஒரு லைப்ரரி மட்டும் தேவைப்படுவதால், அதற்கான கட்டளையை இங்கேயே கொடுத்துவிட்டோம். இதன் பின் நமது ப்ரோக்ராமை இயக்குவதற்குத் தேவையான கட்டளைகள் கொடுக்கப்பட்டுள்ளன.

அடுத்ததாக docker build எனும் கட்டளை மூலம் இமேஜ் உருவாக்கப்பட்டுள்ளது. -t என்பது tag என்பதைக் குறிக்கும். அதாவது sample எனும் பெயர்கொண்ட v1 என tag செய்யப்பட்ட ஒரு இமேஜ் id உருவாக்கப்படும்.

```
$ sudo docker build -t sample:v1 .
```

```
shrini@nithya-Lenovo-ideapad:~/nithya/devops/sample$ sudo docker build -t sample:v1 .
Sending build context to Docker daemon  4.096kB
Step 1/7 : FROM ubuntu:20.04
20.04: Pulling from library/ubuntu
a4a2a29f9ba4: Pull complete
127c9761dcba: Pull complete
d13bf203e905: Pull complete
4039240d2e0b: Pull complete
Digest: sha256:52259450119427dab05c0c455121c48d7b04cee2d61b5dbdde1219b2163af572
Status: Downloaded newer image for ubuntu:20.04
 ---> 74435f89ab78
Step 2/7 : COPY . /app
 ---> d746a6363240
Step 3/7 : WORKDIR /app
 ---> Running in f7be41700f9b
Removing intermediate container f7be41700f9b
 ---> 1d19b28ccd99
Step 4/7 : RUN apt-get update
```

இதைத்தொடர்ந்து real_time எனும் ஃபோல்டருக்குள் சென்று prediction_api எனும் ப்ரோக்ராமை இயக்குவதற்குத் தேவையான டாக்கர் ஃபைலை உருவாக்கவும்.

https://gist.github.com/nithyadurai87/d369ec617fc3dc0dd577bbf08b6913e5

Dockerfile

```
FROM ubuntu:20.04
COPY . /app
WORKDIR /app
RUN apt-get update
RUN apt-get install python3-pip -y
RUN pip3 install -r requirements.txt
ENTRYPOINT ["python3"]
CMD ["prediction_api.py"]
```

requirements.txt

```
pandas
numpy
flask
scikit-learn==0.23.1
joblib
```

இங்கு ப்ரோக்ராமை இயக்குவதற்குத் தேவையான லைப்ரரி அதிகமாக இருப்பதால் அவை அனைத்தும் requirements.txt எனும் கோப்பிற்குள் கொடுக்கப்பட்டு அதிலிருந்து நிறுவுமாறு, டாக்கருக்குள் எழுதப்பட்டுள்ளது.

பின் docker build எனும் கட்டளை மூலம் prediction எனும் பெயர் கொண்ட v1 என tag செய்யப்பட்ட ஒரு இமேஜ் id உருவாக்கப்பட்டுள்ளது.

```
$ sudo docker build -t prediction:v1 .
```

```
Successfully built 79d289e693af
Successfully tagged sample:v1
shrini@nithya-Lenovo-ideapad:~/nithya/devops/sample$ cd ..
shrini@nithya-Lenovo-ideapad:~/nithya/devops$ cd real_time/
shrini@nithya-Lenovo-ideapad:~/nithya/devops/real_time$ ls
data.csv  Dockerfile  model_creation.py  model.pkl  parameters.json  prediction_api.py  pred
iction.py  requirements.txt
shrini@nithya-Lenovo-ideapad:~/nithya/devops/real_time$ sudo docker build -t prediction:v1 .
Sending build context to Docker daemon  225.8kB
Step 1/8 : FROM ubuntu:20.04
 ---> 74435f89ab78
Step 2/8 : COPY . /app
 ---> e88b10380d80
Step 3/8 : WORKDIR /app
```

docker images எனும் கட்டளை இதுவரை நாம் உருவாக்கியுள்ள இமேஜ் அனைத்தையும் பட்டியலிடுவதைக் காணலாம். ஆனால் docker ps எனும் கட்டளை இன்னும் கன்டெய்னர்கள் எதுவும் உருவாக்கப்படவில்லை என்பதைத் தெரிவிக்கிறது.

```
shrini@nithya-Lenovo-ideapad:~/nithya/devops/real_time$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
prediction          v1                  e90ecda6031a        10 seconds ago      708MB
sample              v1                  79d289e693af        13 minutes ago      389MB
ubuntu              20.04               74435f89ab78        2 days ago          73.9MB
shrini@nithya-Lenovo-ideapad:~/nithya/devops/real_time$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
              PORTS               NAMES
```

docker run எனும் கட்டளை கொடுக்கப்பட்ட இமேஜை கன்டெய்னரில் போட்டு, நமது அப்ளிகேஷனுக்கென தனியொரு கன்டெய்னரை உருவாக்கும். டாக்கர் கோப்பில், தேவையான மாடியூலை இன்ஸ்டால் செய்வதற்கான கட்டளைகள் இல்லையெனில் பின்வருவது போன்ற தவறுகள் ஏற்படும்.

```
shrini@nithya-Lenovo-ideapad:~/nithya/devops/real_time$ sudo docker run -i -p 6500:5500 sample:v1
Traceback (most recent call last):
  File "sample_api.py", line 2, in <module>
    from flask import Flask
ModuleNotFoundError: No module named 'flask'
```

இதுபோன்ற தருணத்தில் ஏற்கனவே உருவாக்கிய இமேஜை அழித்து, டாக்கரில் திருத்தம் செய்து மீண்டும் ஒரு இமேஜை உருவாக்க வேண்டும். docker rmi -f என்பது இமேஜை அழிக்க உதவும் கட்டளை ஆகும்.

```
shrini@nithya-Lenovo-ideapad:~/nithya/devops/real_time$ sudo docker rmi -f 79d289e693af
Untagged: sample:v1
Deleted: sha256:79d289e693af1055b51107255bc85b9b5116d37ad9d1d194bd40b3105b5959d5
Deleted: sha256:b13fe7e01bbc717ff3c192a0437a21208774a94bbbaad0ca430e596e7fb110be
Deleted: sha256:9392975fda39bd413d88589147682cc7fbd2a4004451d2153d2d0c446e01569a
Deleted: sha256:4d645a2ab0fcfadde68602ffca28fa034b951ad990e51ce3f1af378221ee7b32
Deleted: sha256:1d19b28ccd998db060edb68ab8c8cb2858c38964884423f7e5bbf5f947cd1f23
Deleted: sha256:d746a6363240efef2c4600cd5e099181ff77274b55fc023ddef1837538a76422
```

அடுத்து திருத்தப்பட்ட இமேஜ் உருவாக்கப்படுகிறது.

```
shrini@nithya-Lenovo-ideapad:~/nithya/devops$ cd sample/
shrini@nithya-Lenovo-ideapad:~/nithya/devops/sample$ sudo docker build -t sample:v2 .
Sending build context to Docker daemon  4.096kB
Step 1/8 : FROM ubuntu:20.04
 ---> 74435f89ab78
Step 2/8 : COPY . /app
 ---> 2b421d1ba7b6
Step 3/8 : WORKDIR /app
 ---> Running in 2f2fb7556fdf
Removing intermediate container 2f2fb7556fdf
 ---> 8fe91c069017
Step 4/8 : RUN apt-get update
 ---> Running in 27e9b271120e
```

இப்போது இரண்டு அப்ளிகேஷன்களுக்குமான இரண்டு கன்டெய்னர்கள் தனித்தனியே பின்வருமாறு உருவாக்கப்பட்டுள்ளன.

```
$ sudo docker run -d -p 6600:5600 sample:v2
$ sudo docker run -d -p 6600:5600 prediction:v1
```

-d என்பது daemon process -ஐக் குறிக்கிறது. இதுவே நமது அப்ளிகேஷனை பின்புறத்திலிருந்து இயங்குமாறு செய்கிறது.

-p என்பது port ஆகும். Colon-க்கு இடப்புறம் ஹோஸ்ட் சர்வரில் மதிப்பு வெளிப்பட வேண்டிய போர்ட் எண்ணையும், வலப்புரம் நம்முடைய லோக்கல் அப்ளிகேஷனுக்குள் நாம் கொடுத்த போர்ட் எண்ணையும் குறிப்பிட வேண்டும்.

```
shrini@nithya-Lenovo-ideapad:~/nithya/devops/sample$
sudo docker run -d -p 6600:5600 prediction:v1
8b9d4379dea8c759ed0b19187e62d96de798c5423b53a9e3ee750
ffb8ad265ba
shrini@nithya-Lenovo-ideapad:~/nithya/devops/sample$
sudo docker run -d -p 6500:5500 sample:v2
fa9e3b4af93a4af2a85e2985e27e3cc5b64356229d1b69cf1ec91
1e5a4653cc9
```

docker ps எனும் கட்டளை நாம் உருவாக்கிய கன்டெய்னர்களை வெளிக்காட்டுவதைக் காணலாம். அவ்வளவுதான்! இந்தக் கன்டெய்னர்கள் வழியே நமது அப்ளிகேஷனுக்கான சர்வீஸ் பின்புலத்தில் இயங்கிக் கொண்டிருக்கும். கொடுக்கப்பட்ட போர்டில் மதிப்புகளை வெளிப்படுத்திக் கொண்டிருக்கும்.

```
shrini@nithya-Lenovo-ideapad:~/nithya/devops/sample$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
fa9e3b4af93a        sample:v2           "python3 sample_api.…"   20 seconds ago
8b9d4379dea8        prediction:v1       "python3 prediction_…"   About a minute ago
```

```
STATUS              PORTS                       NAMES
Up 12 seconds       0.0.0.0:6500->5500/tcp      pensive_bhaskara
Up 59 seconds       0.0.0.0:6600->5600/tcp      infallible_galileo
```

போஸ்ட்மேன் கருவி வழியே இதனை நாம் மீண்டும் சரிபார்த்துக் கொள்ளலாம்.

```
GET  0.0.0.0:6500/api/sample
Authorization  Headers  Body  Pre-request Script  Tests
Type  No Auth
Body  Cookies  Headers (4)  Test Results
Pretty  Raw  Preview  HTML
1 Hello World
```

POST 0.0.0.0:6600/api/salepricemodel

Authorization | Headers (1) | Body | Pre-request Script | Tests

form-data | x-www-form-urlencoded | raw | binary | JSON (application/jsc


```
{
    "OverallQual":[7],
    "TotalBsmtSF":[856],
    "1stFlrSF":[856],
    "GrLivArea":[1710],
    "FullBath":[2],
    "TotRmsAbvGrd":[8],
    "Fireplaces":[0],
    "GarageCars":[2],
    "GarageArea":[548],
    "Years Before Sale":[5],
    "Years Since Remod":[5]
}
```


Body | Cookies | Headers (4) | Test Results

Pretty | Raw | Preview | HTML

```
[211726.49623382]
```

டாக்கர் மூலம் வேறொரு புதிய சர்வரில் ஹோஸ்ட் செய்யாமல் என்னுடைய கணினியையே நான் பயன்படுத்தியதால் இங்கு 0.0.0.0 எனும் லோக்கல் ஹோஸ்ட் இடம்பெற்றுள்ளது. ஆனால் 5500, 5600 ஆகிய போர்ட் எண்களில் இயங்கிக் கொண்டிருந்தது தற்போது 6500, 6600 ஆகிய போர்ட் எண்களில் செயல்படுவது டாக்டரின் இயக்கத்தை உறுதி செய்கிறது.

மேலும் start, stop, rm ஆகிய கட்டளைகள் கன்டெய்னரின் இயக்கத்தை துவக்குதல், நிறுத்துதல், கன்டெய்னரை நீக்குதல் போன்ற வேலைகளைச் செய்கின்றன. முதலில் கன்டெய்னர்களின் நிலைகளையும்(STATUS) இச்செயல்களுக்குப் பின்பு அவற்றின் நிலைகளையும் பின்வருமாறு காணலாம்.

```
shrini@nithya-Lenovo-ideapad:~$ sudo docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
fa9e3b4af93a        sample:v2           "python3 sample_api.…"   4 weeks ago         Exited (0) 6 minutes ago
8b9d4379dea8        prediction:v1       "python3 prediction_…"   4 weeks ago         Up 44 seconds
b748b3551d4b        prediction:v1       "python3 -d"             4 weeks ago         Exited (130) 4 weeks ago
8b4e546206db        prediction:v1       "python3 prediction_…"   4 weeks ago         Exited (0) 4 weeks ago
f30ee1eb308a        sample:v2           "python3 sample_api.…"   4 weeks ago         Exited (0) 4 weeks ago
0b9ed4e68e24        79d289e693af        "python3 sample_api.…"   4 weeks ago         Exited (1) 4 weeks ago
```

```
shrini@nithya-Lenovo-ideapad:~$ sudo docker stop 8b9d4379dea8
8b9d4379dea8
shrini@nithya-Lenovo-ideapad:~$ sudo docker start fa9e3b4af93a
fa9e3b4af93a
shrini@nithya-Lenovo-ideapad:~$ sudo docker rm b748b3551d4b
b748b3551d4b
```

```
shrini@nithya-Lenovo-ideapad:~$ sudo docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
fa9e3b4af93a        sample:v2           "python3 sample_api.…"   4 weeks ago         Up 44 seconds
8b9d4379dea8        prediction:v1       "python3 prediction_…"   4 weeks ago         Exited (0) About a minut
8b4e546206db        prediction:v1       "python3 prediction_…"   4 weeks ago         Exited (0) 4 weeks ago
f30ee1eb308a        sample:v2           "python3 sample_api.…"   4 weeks ago         Exited (0) 4 weeks ago
0b9ed4e68e24        79d289e693af        "python3 sample_api.…"   4 weeks ago         Exited (1) 4 weeks ago
```

ஒரு அப்ளிகேஷனை நிறுவுவதில் டாக்கரின் பயன்பாடு பற்றி இப்பகுதியில் பார்த்தோம். இதில் ஜென்கின்ஸ் பங்களிப்பு பற்றி இனிவரும் பகுதியில் காணலாம்.

7.2 Docker Compose

Develop, Ship & Run multi-container application என்பதே டாக்கர் கம்போஸ்ன் தத்துவம் ஆகும். இதுவரை flask மூலம் ஒரே ஒரு அப்ளிகேஷனை உருவாக்கி, கன்டெய்னரில் இட்டு சர்வரில் deploy செய்வது எப்படி என்று பார்த்தோம். ஆனால் நிஜத்தில் வெறும் அப்ளிகேஷன் மட்டும் உருவாக்கப்படாது. ப்ராஜெக்ட் கட்டமைப்பு என்பது அப்ளிகேஷன், அதற்குரிய டேட்டாபேஸ் என அனைத்தும் சேர்ந்தே வரும். ஆகவே இவை ஒவ்வொன்றுக்கும் தனித்தனி கன்டெய்னரை உருவாக்கி அவற்றை ஒன்றோடொன்று தொடர்பு கொள்ளுமாறு செய்வதே docker-compose ஆகும்.

மேலும் ஒரே கணினியில் மல்டி கன்டெய்னர் அப்ளிகேஷனை இயக்குவது docker-compose என்றால் பல்வேறு கணினிகள் இணைந்திருக்கும் கிளஸ்டர் அமைப்பில் இதே வேலையைச் செய்வது docker swarm ஆகும்.

கீழ்க்கண்ட உதாரண ப்ரோக்ராமில் flask-ன் default port 5000-ல் அளிக்கப்படும் மதிப்பினை எடுத்து மாங்கோ db-ல் செலுத்துமாறு ஒரு அப்ளிகேஷன் எழுதப்பட்டுள்ளது. இதில் இரண்டு கூறுகள் உள்ளன. ஒன்று ஃபிளாஸ்க் மூலம் எழுதப்பட்டுள்ள அப்பிளிகேஷன் மற்றொன்று மாங்கோ db. இவை இரண்டுக்குமான கன்டெய்னர்களை தனித்தனியே எவ்வாறு உருவாக்குவது என்று இப்பகுதியில் பார்க்கலாம்.

https://gist.github.com/nithyadurai87/5617efd02f1e2b3aaef934e11ddac6bb

app.py

```
from flask import Flask, request, jsonify
from pymongo import MongoClient
```

```
app = Flask(__name__)
client = MongoClient('mongodb',27017)
db = client.forest
collection = db.flowers

@app.route('/', methods=['POST', 'GET'])
def index():

    if request.method == 'POST':
        data = request.get_json()
        collection.insert_one(data).inserted_id
        # with open('./tmpfiles/data.log','a') as f:
            # f.write(str(data)+'/n')
        return ('', 204)

    if request.method == 'GET':
        data = collection.find()
        response = []
        for i in data:
            i['_id'] = str(i['_id'])
            response.append(i)
        return jsonify(response)

if __name__ == '__main__':
    app.run(host='0.0.0.0')
```

முதலில் இவற்றுக்கான கன்டெய்னர்களை உருவாக்க எப்போதும் போல் Dockerfile மற்றும் requirements.txt ஆகியவற்றை எழுதிவிடவும்.

Dockerfile

```
FROM ubuntu:20.04
COPY . /app
WORKDIR /app
RUN apt-get update
RUN apt-get install python3-pip -y
RUN pip3 install -r requirements.txt
ENTRYPOINT ["python3"]
CMD ["app.py"]
```

requirements.txt

```
flask
pymongo==3.11.0
```

பின்னர் இவற்றுக்கான கன்டெய்னர்களை தனித்தனியே உருவாக்க docker-compose.yml எனும் ஃபைலை நாம் எழுத வேண்டும்.

https://gist.github.com/nithyadurai87/3a1f0ec44e8b07254320701c8747dbe3

docker-compose.yml

```
version: '3'

services:
  app:
    build: .
    image: flaskapp:v1
    environment:
      - FLASK_ENV=development
    ports:
      - 5000:5000

  mongodb:
    image: mongo
```

இதில் நமது அப்ளிகேஷனில் பயன்படுத்தப்படும் பல்வேறு கூறுகளும் தனித்தனியே வரையறுக்கப்படும். docker-compose எனும் கட்டளை இதில் வரையறுக்கப்பட்டுள்ளதற்கு ஏற்ப தனித்தனியாக இமேஜை உருவாக்கி கன்டெய்னர்களை இயக்கும்.

```
$ docker-compose up
```

```
shrinivasan@shrinivasan-Lenovo-Z50-70:~/nithya/app_stack$ docker-compose up
Creating network "appstack_default" with the default driver
Pulling mongodb (mongo:latest)...
latest: Pulling from library/mongo
5d9821c94847: Pull complete
a610eae58dfc: Pull complete
a40e0eb9f140: Pull complete
3242ba6cef1f: Pull complete
```

இக்கட்டளை அப்ளிகேஷனை பில்டு செய்து இரண்டு தனித்தனி கன்டெய்னர்களில் சர்வீசை இயக்கிக் கொண்டிருக்கும். ctrl+x எனக் கொடுத்தால் சர்வீசை நிறுத்திவிடலாம். ஆனால் துவக்கும் போது -d எனக்கொடுத்து துவக்கினால், அதனை நிறுத்த down எனக் கொடுக்க வேண்டும். ஏனெனில் சர்வீஸ் daemon பிராசஸ் ஆக பின்புலத்தில் இயங்கிக் கொண்டிருக்கும். இதற்கான கட்டளைகள் பின்வருமாறு அமையும்.

```
$ docker-compose up -d
$ docker-compose down
```

இப்போது சர்வீஸ் இயங்கிக் கொண்டிருக்கும் பட்சத்தில் அதனை சோதிக்க curl கட்டளையைப் பயன்படுத்தலாம். முன்பு போஸ்ட்மேன் கருவி வழியே சோதித்துப் பார்த்தோம் அல்லவா! இம்முறை curl-ஐப் பயன்படுத்தலாம்.

5000 எனும் போர்ட்டில் ஒரு json data-வை அளிப்பதற்கான கட்டளை பின்வருமாறு.

```
$ curl --header "Content-Type: application/json" --request
POST --data '{"lotus":10,"tulips":14}' localhost:5000
```

```
shrinivasan@shrinivasan-Lenovo-Z50-70:~/nithya/app_stack$ curl --header "
st POST --data '{"lotus":10,"tulips":14}' localhost:5000
shrinivasan@shrinivasan-Lenovo-Z50-70:~/nithya/app_stack$
```

app.py எனும் நிரலில் கொடுக்கப்பட்டுள்ளது போல இத்தரவினை எடுத்து மாங்கோவில் சென்று forest db-க்குள் flowers collection-க்குள் சேர்ந்து விடும். இதனை சோதிக்க அதே போர்டில் GET செய்து பார்க்கலாம் அல்லது மாங்கோவுக்குள் சென்று பார்க்கலாம்.

மாங்கோவில் சென்று கலெக்சனில் சேமிக்கப்பட்டுள்ளதை எடுத்து 5000 போர்டில் வெளிப்படுத்துவதற்கான கட்டளை பின்வருமாறு.

```
$ curl --request GET localhost:5000
```

```
shrinivasan@shrinivasan-Lenovo-Z50-70:~/nithya/app_stack$ curl localhost:
5000
[
  {
    "_id": "5f686c36854e69772585dd64",
    "lotus": 10,
    "tulips": 14
  }
]
```

மாங்கோவுக்குள்ளேயே சென்று பார்க்க விரும்பினால் கீழ்க்கண்ட கட்டளையை அளிக்கவும்.

```
$ docker exec -it appstack_mongodb_1 /bin/bash
```

இது மாங்கோவை interactive ஷெல்லாக வெளிப்படுத்தும்.

```
$ mongo
```

```
shrinivasan@shrinivasan-Lenovo-Z50-70:~/nithya/app_stack$ docker exec -it appstack_mongodb_1 /bin/bash
root@5d088dd3823f:/# mongo
MongoDB shell version v4.4.1
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("acec19d9-eddc-4924-9b33-c6d29aeed61c") }
MongoDB server version: 4.4.1
```

பின் கீழ்க்கண்ட கட்டளைகளை அளித்து json தரவு சென்று சேர்ந்துள்ளதா என சோதித்துக் கொள்ளலாம்.

```
> show dbs
> use forest
> db.flowers.find()
```

```
> show dbs
admin   0.000GB
config  0.000GB
forest  0.000GB
local   0.000GB
> use forest
switched to db forest
> db.flowers.find()
{ "_id" : ObjectId("5f686c36854e69772585dd64"), "lotus" : 10, "tulips" : 14 }
```

அடுத்ததாக ஒருசில துணைக் கட்டளைகளைப் பற்றிப் பார்ப்போம்.

கீழ்க்கண்ட கட்டளை தனித்தனியே ஓடுகின்ற இரண்டு கன்டெய்னர்களின் பெயர் மற்றும் மற்ற விவரங்களை வெளிக்காட்டும்.

```
$ docker-compose ps
```

```
shrinivasan@shrinivasan-Lenovo-Z50-70:~/nithya/app_stack$ docker-compose ps
       Name                    Command              State    Ports
-------------------------------------------------------------------
appstack_app_1       /bin/sh -c flask run --hos ...   Exit 0
appstack_mongodb_1   docker-entrypoint.sh mongod      Exit 0
```

கீழ்க்கண்ட கட்டளை நெட்வொர்க்குகளை பட்டியலிடும்.

```
$ docker network ls
```

```
shrinivasan@shrinivasan-Lenovo-Z50-70:~/nithya/app_stack$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
e2267b429815        appstack_default    bridge              local
6fd2cda8a500        bridge              bridge              local
36a71385e1eb        docker_default      bridge              local
857cb7a391a8        host                host                local
38a453292d15        nithya_default      bridge              local
45b9a060d87c        none                null                local
```

டாக்கர் கம்போஸ் செய்யும் போதே இரண்டு கன்டெய்னர்களையும் உள்ளடக்கிய ஒரு நெட்வொர்க் உருவாக்கப்படும். இந்த நெட்வொர்க்கில் உள்ள கன்டெய்னர்கள் எந்த ஒரு சிறப்பு அனுமதியும் இன்றி தங்களுக்குள் ஒன்றோடொன்று பேசிக்கொள்ளும். ஆனால் மற்ற நெட்வொர்க்குடன் எந்த ஒரு தொடர்பும் இல்லாமல் தனித்து இயங்கும். இங்கு appstack_default என்ற பெயரில் நெட்வொர்க் உருவாகியுள்ளது. அதாவது நம்முடைய கோப்புகள் எந்த டைரக்டரியில் சேமிக்கப்பட்டுள்ளதோ அந்த டைரக்டரியின் பெயருடன் _default என சேர்த்து நெட்வொர்க்கின் பெயர் உருவாக்கப்படும்.

இது நம்முடைய நெட்வொர்க்கை ஆய்வு செய்ய உதவும்.

```
$ docker network inspect appstack_default
```

```
"ConfigOnly": false,
"Containers": {
    "5d088dd3823fe8f26325375c2478ddcaefbe8c45391696e1a4e5b8c78911
        "Name": "appstack_mongodb_1",
        "EndpointID": "b77dda9c1da05353ed03d8abc9646d4ac8b2681faf
        "MacAddress": "02:42:ac:15:00:02",
        "IPv4Address": "172.21.0.2/16",
        "IPv6Address": ""
    },
    "671ac2fc2909d4390060c0fe637ffc4e08011cb9f2450d2f96d11603a561
        "Name": "appstack_app_1",
        "EndpointID": "dfc348e0668689a061cf5fcf75edb05e51785b5b18
        "MacAddress": "02:42:ac:15:00:03",
        "IPv4Address": "172.21.0.3/16",
        "IPv6Address": ""
    }
```


docker-compose ஐத் தொடர்ந்து ps, logs, start, stop, kill, restart, push, pull போன்ற பல்வேறு துணைக் கட்டளைகளையும் பயன்படுத்தலாம்.

7.3 Docker Volume

கீழ்க்கண்ட உதாரணத்தில் என்னென்ன தரவுகள் மங்கோவிற்குள் செலுத்தப்பட்டன என்பதை ஒரு log ஃபைல் போன்று சேமிப்பதற்கான நிரல் இணைக்கப்பட்டுள்ளது. ஆனால் இக்கோப்பு கன்டெய்னருக்குள்ளேயே சேமிக்கப்படும். கண்டெய்னர் தனது இயக்கத்தை நிறுத்தும் போது இதுவும் அழிந்துவிடும் அபாயம் உள்ளது. ஆகவே கன்டெய்னருக்குள்ளேயே சேமிக்கப்படும் இதுபோன்ற தரவுகளை வெளியே எடுத்து லோக்கலில் அணுகுவதற்கு volume என்ற ஒன்று பயன்படுகிறது. இதனைக் கையாள்வது பற்றி கம்போஸ் ஃபைலில் பார்க்கலாம்.

https://gist.github.com/nithyadurai87/5617efd02f1e2b3aaef934e11ddac6bb

```
from flask import Flask, request, jsonify
from pymongo import MongoClient

app = Flask(__name__)
client = MongoClient('mongodb',27017)
db = client.forest
collection = db.flowers

@app.route('/', methods=['POST', 'GET'])
def index():

    if request.method == 'POST':
        data = request.get_json()
        collection.insert_one(data).inserted_id
        with open('./tmpfiles/data.log','a') as f:
            f.write(str(data)+'/n')
        return ('', 204)

    if request.method == 'GET':
        data = collection.find()
        response = []
        for i in data:
            i['_id'] = str(i['_id'])
            response.append(i)
        return jsonify(response)

if __name__ == '__main__':
    app.run(host='0.0.0.0')
```

டாக்கர் கம்போஸ் ஃபைலில் volumes என ஒரு பகுதியைக் குறிப்பிட வேண்டும். பின் அதன்கீழ் கண்டெய்னருக்குள் தரவுகள் சேமிக்கப்படும் பகுதியானது, நம் கணினியில் ஒரு லோக்கல் பாதையுடன் mount செய்யப்படுகிறது. எனவே ஒவ்வொரு முறை கண்டெய்னருக்குள் தரவுகள் சென்று சேரும்போதும், அவை இப்பகுதியிலும் சேமிக்கப்படுகின்றன.

https://gist.github.com/nithyadurai87/3a1f0ec44e8b07254320701c8747dbe3

docker-compose.yml

```
version: '3'

services:
  app:
    build: .
    image: flaskapp:v10
    environment:
      - FLASK_ENV=development
    ports:
      - 5000:5000
    volumes:
      - /home/nithya/backup:/app/tmpfiles

  mongodb:
    image: mongo
```

இப்போது app.py மற்றும் docker-compose.yml ஆகிய கோப்புகளில் மாற்றம் செய்துள்ளதால் மீண்டும் ஒருமறை அவற்றுக்கான இமேஜ் உருவாக்கப்படுகிறது.

```
shrinivasan@shrinivasan-Lenovo-Z50-70:~/nithya/app_stack$ ls
Dockerfile  app.py  docker-compose.yml  requirements.txt
shrinivasan@shrinivasan-Lenovo-Z50-70:~/nithya/app_stack$ docker-compose up -d
Building app
Step 1/6 : FROM ubuntu:20.04
 ---> 9140108b62dc
Step 2/6 : COPY . /app
 ---> 89edd2bcf8ae
Step 3/6 : WORKDIR /app
 ---> Running in 8d1ff2fc5938
Removing intermediate container 8d1ff2fc5938
 ---> bbb51dc652c1
```

அடுத்து கீழ்க்கண்ட தரவு மாங்கோவிற்குள் செலுத்தப்படுகிறது.

```
$ curl --header "Content-Type: application/json" --request
POST --data '{"lotus":1033333,"tulips":166664}'
localhost:5000
```

இத்தரவு கன்டெய்னருக்குள் data.log என்ற பெயரில் சேமிக்கப்படுவதைக் காணலாம். app.py இயங்கிக் கொண்டிருக்கும் கண்டெய்னருக்குள் செல்ல அதன் தொடக்க எண்ணான 21 என்பது கொடுக்கப்பட்டுள்ளது. இந்த எண்ணானது $ docker ps எனும் கட்டளை மூலம் கண்டறியப்படுகிறது.

```
shrinivasan@shrinivasan-Lenovo-Z50-70:~/nithya/app_stack$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
                   NAMES
2149cd3a4d98        flaskapp:v10        "python3 app.py"         About a
0:5000->5000/tcp   appstack_app_1
f28c42366764        mongo               "docker-entrypoint.s…"   2 hours
tcp                appstack_mongodb_1
```

```
$ docker exec -it 21 /bin/sh
# cat tmpfiles/data.log
```

```
shrinivasan@shrinivasan-Lenovo-Z50-70:~/nithya/app_stack$ docker exec -it 21 /bin/sh
# ls
Dockerfile  app.py  docker-compose.yml  requirements.txt  tmpfiles
# cd tmpfiles
# ls
data.log
# cat data.log
{'lotus': 1033333, 'tulips': 166664, '_id': ObjectId('5f730f085288654948a97a59')}/n#
```

மேலும் இதே தரவு நம்முடைய லோக்கல் கணினியில் /home/nithya/backup எனுமிடத்தில் சேமிக்கப்படுவதையும் காணலாம்.

```
shrinivasan@shrinivasan-Lenovo-Z50-70:/home/nithya/backup$ cat data.log
{'lotus': 1033333, 'tulips': 166664, '_id': ObjectId('5f730f085288654948a97a59')}/
```

இப்பகுதியில் docker, docker-compose, docker volumes ஆகியவற்றின் பயன்பாடுகளைப் பற்றியெல்லாம் பார்த்தோம்.. அடுத்ததாக ஜென்கின்ஸ் கொண்டு இந்த டாக்கர் இமேஜை எவ்வாறு தானியக்க முறையில் உருவாக்குவது என்று பார்க்கலாம்.

8. Jenkins

ஒரு மென்பொருள் உருவாக்கத்தின் பல்வேறு நிலைகளான அப்ளிகேஷனின் உருவாக்கம், சோதனை, பல்வேறு சர்வர்களில் நிறுவுதல் போன்ற வெவ்வேறு தனித்தனி செயல்களை தானியக்க முறையில் தொடர்ச்சியாக நிகழ்த்த உதவும் கருவியே ஜென்கின்ஸ் என்று அழைக்கப்படுகிறது. எனவேதான் இது தொடர்ச்சியான ஒருங்கிணைப்பு(CI) மற்றும் தொடர்ச்சியான வழங்குதலுக்கான(CD) கருவி என்று அழைக்கப்படுகிறது. டெவலப்பர் ஒவ்வொருமுறை மூல நிரலில் மாற்றம் செய்து கமிட் செய்யும்போதும், அதற்கான அப்ளிகேஷனை சுலபமாக சர்வரில் நிறுவி சோதித்துப் பார்க்க உதவும் ஒரு கருவியாக ஜென்கின்ஸ் விளங்குகிறது. Git, SVN போன்றவற்றுடன் இது ஒருங்கிணைந்து செயல்படுகிறது.

நம்முடைய கிட் சர்வரில் real_time எனும் ஃபோல்டருக்குள் உள்ள அப்ளிகேஷனை ஜென்கின்ஸ் மூலம் எவ்வாறு சர்வரில் நிறுவுவது என்று இப்பகுதியில் பார்க்கலாம்.

ஜென்கின்ஸ் நிறுவுவற்கான மற்றும் நீக்குவதற்கான கட்டளைகள் பின்வருமாறு.

```
sudo apt-get install jenkins
sudo apt-get remove jenkins
```

அதனை துவக்குவதற்கான மற்றும் நிறுத்துவதற்கான கட்டளைகள் பின்வருமாறு.

```
sudo systemctl start jenkins
sudo systemctl stop jenkins
```

ஜென்கின்ஸ் தற்போது எந்த நிலையில் உள்ளது என்பதை அறிய பின்வரும் கட்டளை பயன்படுகிறது.

```
sudo systemctl status jenkins
```

```
shrini@nithya-Lenovo-ideapad:~$ sudo systemctl status jenkins
● jenkins.service - LSB: Start Jenkins at boot time
     Loaded: loaded (/etc/init.d/jenkins; generated)
     Active: active (exited) since Fri 2020-07-17 16:25:10 IST
       Docs: man:systemd-sysv-generator(8)
      Tasks: 0 (limit: 9282)
     Memory: 0B
     CGroup: /system.slice/jenkins.service
```

மேற்கண்ட கட்டளை active என்று வெளிப்படுத்தினால் இதற்கான சர்வர் பின்வரும் முகவரியில் இயங்கிக் கொண்டிருக்கும்.

http://localhost:8080/login

முதன்முதலில் இதற்குள் நுழைவதற்குத் தேவையான சான்றுகளை எவ்வாறு பெறுவது என்பது பற்றி பின்வரும் இணைப்பில் விளக்கமாக காணலாம்.
https://www.digitalocean.com/community/tutorials/how-to-install-jenkins-on-ubuntu-18-04

பின் நமக்கான பயனர்பெயர் மற்றும் கடவுச்சொல்லைப் பயன்படுத்தி உள்நுழைந்து கொள்ளலாம்.

இப்போது நமக்கான price_prediction எனும் பெயர்கொண்ட ஒரு ப்ராஜெக்டை உருவாக்க New Item எனும் இணைப்பின் மீது சொடுக்கி, பெயர் அளித்து Freestyle Project என்பதை தேர்வு செய்து OK எனும் பொத்தானின் மீது சொடுக்கவும்.

நமக்கான ப்ராஜெக்ட் உருவாக்கப்பட்ட பின் அதில் பின்வரும் ஆறு பகுதிகள் இருப்பதைக் காணலாம்.

General
Source Code Management
Build Triggers
Build Environment
Build
Post-build Actions

இவற்றில் மூல நிரல் மேலாண்மைக்கான பகுதியில் சென்று Git என்பதை தேர்வு செய்து நாம் உருவாக்கிய கிட் சர்வரின் முகவரியைக் கொடுக்கவும்.

General | **Source Code Management** | Build Triggers | Build Environment | Build | Post-build Actions

Source Code Management

- None
- Git

Repositories

Repository URL: https://github.com/nithyadurai87/devops_examples

Credentials: - none - | Add

Advanced...

Add Repository

அடுத்து அப்ளிகேஷனை எவ்வாறு நிறுவ வேண்டும் என்பதற்கான கட்டளைகளை ஒன்றன்பின் ஒன்றாக Build எனும் பகுதியில் நாம் குறிப்பிட வேண்டும்.

Build

Add build step

- Execute Windows batch command
- Execute shell
- Invoke Ant
- Invoke Gradle script
- Invoke top-level Maven targets
- Run with timeout
- Set build status to "pending" on GitHub commit

Build

Execute shell

Command

```
rm -rf ./*
git clone https://github.com/nithyadurai87/devops_examples
cd devops_examples/real_time
docker build -t prediction:${BUILD_NUMBER} .
```

See the list of available environment variables

Advanced...

கீழ்க்கண்ட கட்டளைகள் தொடர்ச்சியாக இங்கே கொடுக்கப்பட்டுள்ளது. அதாவது தற்போதைய டைரக்டரியில் இருக்கும் அனைத்தையும் அழித்து புத்தம்புதிதாக கிட் ரெப்பாசிட்டரியை பதிவிறக்கம் செய்து, பின்னர் டாக்கர் இமேஜை உருவாக்குவதற்கான கட்டளைகள் இவை.

```
rm -rf ./*
git clone https://github.com/nithyadurai87/devops_examples
cd devops_examples/real_time
docker build -t prediction:${BUILD_NUMBER} .
```

இவற்றை அளித்தபின் Save-ன் மீது சொடுக்கவும். இப்போது நமது அப்ளிகேஷனுக்கான டாக்டர் இமேஜை உருவாக்கக்கூடிய ஜென்கின்ஸ் ப்ராஜெக்ட் அமைக்கப்பட்டு விட்டது. இதைத்தொடர்ந்து இடது பகுதியில் உள்ள Build Now இணைப்பின் மீது சொடுக்கவும்.

Jenkins › price_prediction ›

Back to Dashboard
Status
Changes
Workspace
Build Now
Delete Project
Configure
Rename

Project price_prediction

Workspace
Recent Changes

Permalinks

Build History trend

கொடுக்கப்பட்ட கட்டளைகளின் படி இமேஜை உருவாக்கும் வேலையை இது செய்யத் தொடங்கிவிடும். Build History -ல் உள்ள #1 என்பது இதனுடைய ரன் எண் ஆகும். அதாவது முதல் முறையாக இதற்கான இமேஜை பில்டு செய்து கொண்டிருக்கிறது என்று அர்த்தம்.

Configure

Rename

Build History trend

find x

#1

Fri Jul 24 20:01:27 IST 2020

Atom feed for all Atom feed for failures

இந்த எண்ணின் மீது வலம் சொடுக்கி கன்சோல் அவுட்புட் என்பதைத் தேர்வு செய்தால் கட்டளைகளின் இயக்கங்களைத் திரையில் காணலாம்.

Jenkins › price_prediction › #1

Back to Project

Status

Changes

Console Output

View as plain text

Edit Build Information

Delete build '#1'

Git Build Data

Console Output

```
Started by user Nithya Duraisamy
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/price_prediction
No credentials specified
Cloning the remote Git repository
Cloning repository https://github.com/nithyadurai87/devops_examples
 > git init /var/lib/jenkins/workspace/price_prediction # timeout=10
Fetching upstream changes from https://github.com/nithyadurai87/devops_
 > git --version # timeout=10
 > git fetch --tags --force --progress -- https://github.com/nithyadur
```

```
Step 7/8 : ENTRYPOINT ["python3"]
 ---> Running in 894626f9b1bc
Removing intermediate container 894626f9b1bc
 ---> bc59d07f6073
Step 8/8 : CMD ["prediction_api.py"]
 ---> Running in f19eb9cb28d2
Removing intermediate container f19eb9cb28d2
 ---> 557e786238fe
Successfully built 557e786238fe
Successfully tagged prediction:1
Finished: SUCCESS
```

கன்சோலின் கடைசியில் சக்சஸ் என வெளிப்பட்டுள்ளது இமேஜ் வெற்றிகரமாக உருவாக்கப்பட்டுள்ளதை உணர்த்துகிறது. நாம் பில்டு பகுதியில் குறிப்பிட்டுள்ளது prediction:${BUILD_NUMBER} போலவே prediction:1 எனும் இமேஜ் உருவாக்கப்பட்டது. இதனை டெர்மினலில் சோதித்துப் பார்ப்பது பின்வருமாறு.

```
$ sudo docker images -a
```

```
shrini@nithya-Lenovo-ideapad:~$ sudo docker images -a
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
prediction          1                   557e786238fe        18 hours ago        670MB
```

இப்பகுதியில் நமது அப்ளிகேஷனுக்குத் தேவையான டாக்கர் இமேஜை எவ்வாறு உருவாக்குவது என்று மட்டும் பார்த்தோம். இதனோடு சேர்த்து டாக்கர் இமேஜை ரன் செய்வதற்கான கட்டளையையும் ஜென்கின்ஸின் பில்டு பகுதியில் கொடுத்திருந்தால் இந்நேரம் கன்டெய்னர் ஓடிக்கொண்டிருக்கும். ஆனால் நிஜத்தில் இதுபோன்று நடக்காது. டாக்கர் பில்டு மற்றும் டாக்கர் ரன் ஆகிய இரண்டு கட்டளைகளையும் ஒருசேர இயக்க முடியாது. ஏனெனில் ஜென்கின்ஸ் ஒரு சர்வரில் ஓடிக் கொண்டிருக்கும். நமது அப்ளிகேஷனுக்குத் தேவையான டாக்கர் மற்றொரு சர்வரில் ஓடவேண்டி வரும். ஆகவே வெறும் இமேஜை மட்டும் உருவாக்கி, அதனை Docker Hub எனும் மைய சர்வருக்கு அனுப்பிவிடலாம். பின் அங்கிருந்து நமக்குத் தேவையான சர்வரில் சென்று இமேஜை டவுன்லோட் செய்து டாக்கரை ரன் செய்யலாம். நிஜத்தில் டாக்கர் மற்றும் ஜென்கின்ஸ் பயன்பாடு இவ்வாறுதான் இருக்கும்.

ஒரு அப்ளிகேஷனை தானியக்க முறையில் ஜென்கின்ஸ் துணைகொண்டு எவ்வாறு நிறுவுவது என்று இப்பகுதியில் பார்த்தோம்.. அடுத்ததாக, எங்கிருந்தோ தொடர்ச்சியாக வரும் செய்திகளை வாங்கி வாங்கி மற்றோரிடத்திற்கு அனுப்ப உதவும் Kafka கருவியைப் பற்றிக் கொஞ்சம் தெரிந்து கொள்வோம்.

9. Kafka

நிகழ் நேரத்தில் அதிக அளவு உற்பத்தியாகும் (throughput) தரவு ஊட்டங்களை (data feed) குறைந்த காலதாமதத்தில் (low latency) பெற்று ப்ராசஸ் செய்வதற்கான ஒரு கட்டமைப்பே kafka ஆகும். இது scala மொழியில் எழுதப்பட்ட திறந்த மூல மென்பொருள் கருவி ஆகும். ப்ரொடியூசர் கன்ஸ்யூமர் என்னும் இருவேறு அப்ளிகேஷன்களுக்கு இடையே செய்திகளைத் தாங்கிச் செல்லும் இடைத்தரகர் போன்று இக்கருவி செயல்படும். IOT சென்சார் தரவுகள், சேவை மையங்களில் தினசரி மேற்கொள்ளப்படும் தொலைபேசி அழைப்புகள், ஒரு வங்கியின் பல்வேறு ஏடிஎம் அட்டைகளில் பல்வேறு இடங்களில் மேற்கொள்ளப்படும் பணப் பரிமாற்றங்கள் போன்றவையெல்லாம் ப்ரொடியூசராக இயங்கி தரவுகளை வழங்கிக் கொண்டே இருந்தால் அவற்றையெல்லாம் சேமிக்க உதவும் data lake, hdfs, mongodb போன்ற nosql டேட்டாபேஸ் ஆகியவை கன்ஸ்யூமராக இயங்கி அவற்றைப் பெற்றுக் கொள்ளும். இவ்விரண்டுக்கும் இடையே topic என்ற ஒன்றை உருவாக்கி அதன் கீழ் உடனுக்குடன் தரவுகளைப் பகிரும் வேலையை kafka செய்கிறது.

தரவுகளை வழங்குவது ப்ரொடியூசர் எனவும், பெற்றுக்கொள்வது கன்ஸ்யூமர் எனவும் அழைக்கப்படும். இவ்விரண்டுக்கும் இடையில் ஒவ்வொரு partition-ஆக தரவுகளின் பகிர்வு நடைபெறுவதைக் கண்காணிக்கும் வேலையை zoo keeper செய்கிறது. ஜூ கீப்பர் இல்லாமல் கஃப்காவை நாம் இயக்க முடியாது.

கஃப்காவை கீழ்க்கண்ட முகவரியில் சென்று பதிவிறக்கம் செய்து பிரித்தெடுத்து வைத்துக் கொள்ளவும்.

```
$ wget "https://downloads.apache.org/kafka/2.5.0/kafka_2.12-2.5.0.tgz"

$ tar -xzf kafka_2.12-2.5.0.tgz
$ cd kafka_2.12-2.5.0
```

```
shrini@nithya-Lenovo-ideapad:~$ wget "https://downloads.apache.org/kafka/2.5.0/kafka_2.12-2.5.0.tgz"
--2020-07-18 11:50:01--  https://downloads.apache.org/kafka/2.5.0/kafka_2.12-2.5.0.tgz
Resolving downloads.apache.org (downloads.apache.org)... 88.99.95.219, 2a01:4f8:10a:201a::2
Connecting to downloads.apache.org (downloads.apache.org)|88.99.95.219|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 61604633 (59M) [application/x-gzip]
Saving to: 'kafka_2.12-2.5.0.tgz'

kafka_2.12-2.5.0.tg 100%[===================>]  58.75M  1.96MB/s    in 69s

2020-07-18 11:51:12 (867 KB/s) - 'kafka_2.12-2.5.0.tgz' saved [61604633/61604633]

shrini@nithya-Lenovo-ideapad:~$ tar -xzf kafka_2.12-2.5.0.tgz
shrini@nithya-Lenovo-ideapad:~$ cd kafka_2.12-2.5.0
```

9.1 Zoo Keeper & Kafka Server

கஃப்கா மற்றும் ஜூ கீப்பர் ஆகிய இரண்டுக்குமான சர்வர்களை இரண்டு தனித்தனி டெர்மினல்களில் ஓடவிட வேண்டும். இது பின்வருமாறு.

Terminal1:

```
$ bin/zookeeper-server-start.sh config/zookeeper.properties
```

```
shrini@nithya-Lenovo-ideapad:~/kafka_2.12-2.5.0$ ls
bin  config  libs  LICENSE  logs  NOTICE  site-docs
shrini@nithya-Lenovo-ideapad:~/kafka_2.12-2.5.0$ bin/zookeeper-server-start.sh config/zookeeper.properties
[2020-07-18 11:58:47,865] INFO Reading configuration from: config/zookeeper.properties (org.apache.zookeepe
[2020-07-18 11:58:47,867] WARN config/zookeeper.properties is relative. Prepend ./ to indicate that you're
[2020-07-18 11:58:47,872] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.Quorum
[2020-07-18 11:58:47,872] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerCo
[2020-07-18 11:58:47,874] INFO autopurge.snapRetainCount set to 3 (org.apache.zookeeper.server.DatadirClean
[2020-07-18 11:58:47,874] INFO autopurge.purgeInterval set to 0 (org.apache.zookeeper.server.DatadirCleanup
[2020-07-18 11:58:47,874] INFO Purge task is not scheduled. (org.apache.zookeeper.server.DatadirCleanupMana
[2020-07-18 11:58:47,874] WARN Either no config or no quorum defined in config, running  in standalone mode
[2020-07-18 11:58:47,875] INFO Log4j found with jmx enabled. (org.apache.zookeeper.jmx.ManagedUtil)
```

Terminal2:

```
$ bin/kafka-server-start.sh config/server.properties
```

```
shrini@nithya-Lenovo-ideapad:~/kafka_2.12-2.5.0$ bin/kafka-server-start.sh config/server.properties
[2020-07-18 12:21:28,144] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jC
[2020-07-18 12:21:28,679] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiation=true to disable
9Util)
[2020-07-18 12:21:28,743] INFO Registered signal handlers for TERM, INT, HUP (org.apache.kafka.commo
[2020-07-18 12:21:28,749] INFO starting (kafka.server.KafkaServer)
[2020-07-18 12:21:28,750] INFO Connecting to zookeeper on localhost:2181 (kafka.server.KafkaServer)
```

இவை இயங்குவதில் ஏதேனும் பிரச்சினை ஏற்பட்டால் server. properties எனும் கோப்பிற்குள் சென்று listeners என்பது கமெண்ட் செய்யப்பட்டிருந்தால் அதனை நீக்கிவிடவும். பின் அதன் மதிப்பினை PLAINTEXT://localhost:9092 என அமைக்கவும்.

```
$ vim config/server.properties
listeners =  PLAINTEXT://localhost:9092
```

```
shrini@nithya-Lenovo-ideapad:~/kafka_2.12-2.5.0$ ls
bin  config  libs  LICENSE  logs  NOTICE  site-docs
shrini@nithya-Lenovo-ideapad:~/kafka_2.12-2.5.0$ vim config/server.properties
```

```
############################# Socket Server Settings ####

# The address the socket server listens on. It will get t
# java.net.InetAddress.getCanonicalHostName() if not conf
#   FORMAT:
#     listeners = listener_name://host_name:port
#   EXAMPLE:
#     listeners = PLAINTEXT://your.host.name:9092
listeners=PLAINTEXT://localhost:9092
```

9.2 Topic creation

மேற்கண்ட இரண்டும் வெவ்வேறு டெர்மினல்களில் ஓடிக்கொண்டிருக்கும் போது புதிதாக ஒரு டெர்மினலை உருவாக்கி, அதில் தரவுகளைப் பகிர்வதற்குத் தேவையான ஒரு டாப்பிக்கை உருவாக்கவும். இங்கு கணியம் எனும் பெயரில் ஒரு டாப்பிக் உருவாக்கப்பட்டுள்ளது. அடுத்துள்ள கட்டளை இதுவரை நாம் உருவாக்கியுள்ள டாப்பிக் அனைத்தையும் பட்டியலிட உதவும்.

Terminal3:

```
$ bin/kafka-topics.sh --create --bootstrap-server
localhost:9092 --replication-factor 1 --partitions 1 --topic
kaniyam
```

```
shrini@nithya-Lenovo-ideapad:~/kafka_2.12-2.5.0$ bin/kafka-topics.sh --create --bootstrap-server localhost:90
92 --replication-factor 1 --partitions 1 --topic kaniyam
Created topic kaniyam.
shrini@nithya-Lenovo-ideapad:~/kafka_2.12-2.5.0$ bin/kafka-topics.sh --list --bootstrap-server localhost:9092
kaniyam
```

9.3 Producer - Consumer Model

ஓரிடத்திலிருந்து தகவல்களை வழங்குவது producer எனவும், அதனை மற்றொரு இடத்திலிருந்து பெற்றுக் கொள்வது consumer எனவும் அழைக்கப்படும். மிக எளிமையாக இதனை செய்து பார்க்க ஒரு டெர்மினலில் ப்ரொடியூசரை இயக்கி மற்றொரு டெர்மினலில் கன்ஸ்யூமரை இயக்கவும். இந்த டெர்மினலில் நாம் கொடுக்கின்ற தரவுகள் அனைத்தும் அடுத்த டெர்மினலில் உடனுக்குடன் வெளிப்படுகிறதா என சோதிக்கலாம்.

```
$ bin/kafka-console-producer.sh --bootstrap-server
localhost:9092 --topic kaniyam
```

It’s a website for open-source technologies.

இப்படி ஒவ்வொரு வரியாக புரொடியூசர் இயங்குகின்ற இடத்தில் மெசேஜை அளிக்கவும்.

```
shrini@nithya-Lenovo-ideapad:~/kafka_2.12-2.5.0$ bin/kafka-console-producer.sh
--bootstrap-server localhost:9092 --topic kaniyam
>It’s a website for open-source technologies.
>You can find articles in tamil here.
>You can also contribute
>Nice to be part of such communities like this
```

கன்ஸ்யூமர் இயங்குகின்ற இடத்தில் சென்று கொடுக்கப்பட்ட வரிகள் ஒவ்வொரு முறையும் உடனுக்குடன் வெளிப்படுகிறதா என சோதிக்கவும்.

```
$ bin/kafka-console-consumer.sh --bootstrap-server
localhost:9092 --topic kaniyam --from-beginning
```

```
shrini@nithya-Lenovo-ideapad:~/kafka_2.12-2.5.0$ bin/kafka-console-consumer.sh
--bootstrap-server localhost:9092 --topic kaniyam --from-beginning
It’s a website for open-source technologies.
You can find articles in tamil here.
You can also contribute
Nice to be part of such communities like this
```

சோதித்து முடித்தபின் கன்ஸ்யூமரையோ புரொடியூசரையோ ctrl+d எனக் கொடுத்து நிறுத்தினால் மொத்தம் எத்தனை வரிகள் பகிரப்பட்டுள்ளது என்பதை வெளிப்படுத்தும். இங்கு Processed a total of 4 messages என்பதை வெளிப்படுத்தியுள்ளது.

9.4 Multinode Cluster

ஒரே கணினியில் பல்வேறு nodes கொண்ட கிளஸ்டரை நாம் உருவாக்கலாம். இதற்காக பல்வேறு சர்வர்களைப் பயன்படுத்துவதற்கு பதிலாக ஒரே சர்வரில் பல்வேறு nodes-ஐ இயக்குவதன் மூலம் இது சாத்தியமாகிறது. கஃப்காவின் config போஃல்டருக்குள் நமது லோக்கல் சர்வருடைய பண்புகளைப் பெற்று விளங்கும் கோப்பினைக் காணலாம். இதனை அப்படியே நகலெடுத்து அவற்றை சர்வர்1, சர்வர்2 எனும் பெயரில் சேமிக்கவும்.

```
$ cp server.properties server1.properties
$ cp server.properties server2.properties
```

```
shrini@nithya-Lenovo-ideapad:~/kafka_2.12-2.5.0/config$ ls
connect-console-sink.properties    log4j.properties
connect-console-source.properties  producer.properties
connect-distributed.properties     server1.properties
connect-file-sink.properties       server2.properties
connect-file-source.properties     server.properties
connect-log4j.properties           tools-log4j.properties
connect-mirror-maker.properties    trogdor.conf
connect-standalone.properties      zookeeper.properties
consumer.properties
```

பின் இத்தகைய கோப்புகளில் உள்ள சில முக்கியப் பண்புகளின் மதிப்புகளை பின்வருமாறு அமைப்பதன் மூலம் மூன்று ப்ரோக்கர்களை மூன்று வெவ்வேறு port-இல் இயங்குமாறு செய்யலாம்.

```
$ vim server.properties
broker.id=0
listeners=PLAINTEXT://localhost:9092
log.dirs=/tmp/kafka-logs
```

```
$ vim server1.properties
broker.id=1
listeners=PLAINTEXT://localhost:9093
log.dirs=/tmp/kafka-logs-1
```

```
$ vim server2.properties
broker.id=2
listeners=PLAINTEXT://localhost:9094
log.dirs=/tmp/kafka-logs-2
```

ஏற்கனவே கஃப்கா மற்றும் ஜூ கீப்பர் ஆகிய இரண்டுக்குமான சர்வர்கள் இரண்டு தனித்தனி டெர்மினல்களில் ஓடிக்கொண்டிருக்க இப்போது புதிதாக உருவாக்கிய இரண்டு சர்வர்களையும் இரண்டு டெர்மினல்களில் ஓடவிட வேண்டும்.

Terminal4:

```
$ bin/kafka-server-start.sh config/server1.properties
```

Terminal5:

```
$ bin/kafka-server-start.sh config/server2.properties
```

இப்போது மூன்று சர்வர்கள் மூன்று port-இல் ஓடிக்கொண்டிருக்கும் நிலையில், 9092 எனும் போர்டில் carrot எனும் டாப்பிக்கை உருவாக்கவும். அப்போது 3 இடங்களில் பிரதி எடுத்து வைக்குமாறு கொடுக்கவும். எனவே ஒரு இடத்தில் இது அழிக்கப்பட்டாலும், மற்ற இடங்களில் இதன் செயல்பாட்டைக் காணலாம்.

Terminal6:

```
$ bin/kafka-topics.sh --create --bootstrap-server
localhost:9092 --replication-factor 3 --partitions 1 --topic
carrot
```

```
shrini@nithya-Lenovo-ideapad:~/kafka_2.12-2.5.0$ bin/kafka-topics.sh --describe --bootstrap-server
 localhost:9092 --topic carrot
Topic: carrot   PartitionCount: 1       ReplicationFactor: 3    Configs: segment.bytes=1073741824
        Topic: carrot   Partition: 0    Leader: 0       Replicas: 0,2,1 Isr: 0,2,1
shrini@nithya-Lenovo-ideapad:~/kafka_2.12-2.5.0$ bin/kafka-topics.sh --describe --bootstrap-server
 localhost:9092 --topic kaniyam
Topic: kaniyam  PartitionCount: 1       ReplicationFactor: 1    Configs: segment.bytes=1073741824
        Topic: kaniyam  Partition: 0    Leader: 0       Replicas: 0     Isr: 0
```

9092 எனும் போர்டில் ஓடிக்கொண்டிருக்கும் சர்வரின் பிராசஸ் எண்ணைக் கண்டுபிடித்து அழிக்க கீழ்கண்ட கட்டளைகள் பயன்படுகின்றன.

```
$ ps aux | grep server.properties
$ kill -9 19042
```

அதன்பின் carrot எனும் டாப்பிக் 9093 எனும் போர்டில் இடம்பெற்றிருப்பதைக் காணலாம். இதுவே multi-node cluster-க்கு சிறந்த உதாரணமாக அமையும்.

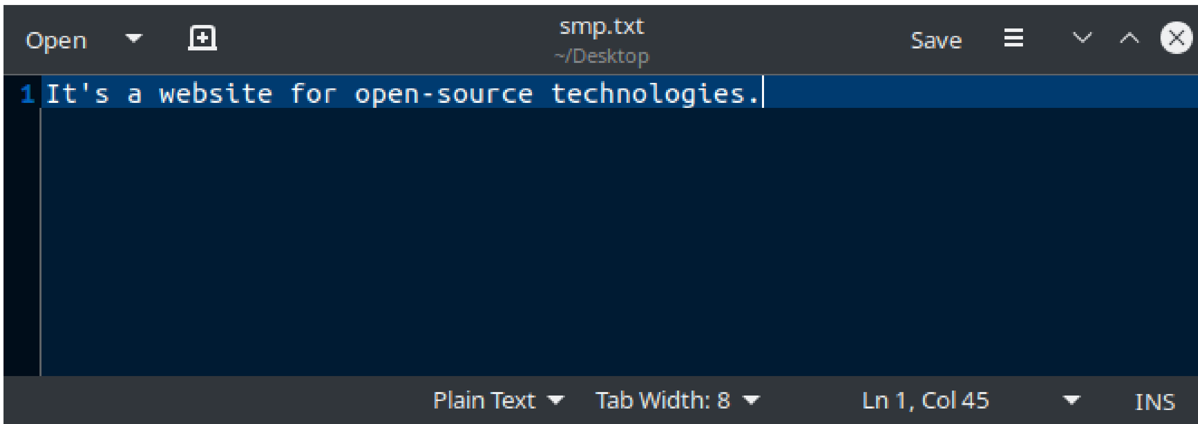
```
$ bin/kafka-topics.sh --describe --bootstrap-server
localhost:9093 --topic carrot
```

```
^Cshrini@nithya-Lenovo-ideapad:~/kafka_2.12-2.5.0$ bin/kafka-topics.sh --describe --bootstrap-serv
er localhost:9093 --topic carrot
Topic: carrot   PartitionCount: 1       ReplicationFactor: 3    Configs: segment.bytes=1073741824
        Topic: carrot   Partition: 0    Leader: 2       Replicas: 0,2,1 Isr: 2,1
```

9.5 File Streaming: kafka-connect

பல்வேறு மூலங்களில் இருக்கும் தரவுகளை கஃப்கா டாப்பிக் வழியே இன்னபிற இடங்களில் செலுத்துவதற்கு kafka-connect எனும் கருவி பயன்படுகிறது. தொடர்ச்சியாக ஓரிடத்தில் கோப்புகள் வந்து விழும்போது அவற்றை எடுத்து ஒரு டாப்பிக்குக்குள் செலுத்த FileSource connector பயன்படுகிறது. அவ்வாறே கஃப்கா டாப்பிக்குக்குள் இருப்பவற்றை ஒரு ஃபைலில் எழுதி ஓரிடத்தில் சேமிக்க FileSink connector பயன்படுகிறது.

கீழ்கண்ட எடுத்துக்காட்டில் smp.txt எனும் கோப்பிற்குள் உள்ளவற்றை கணியம் எனும் டாப்பிக்குக்குள் செலுத்துவதற்கும், பின்னர் அந்த டாப்பிக்குக்குள் உள்ளவற்றை sample.txt எனும் கோப்பினுள் செலுத்துவதற்குமான connectors பின்வருமாறு அமையும்.



Config போஃல்டருக்குள் உள்ள இத்தகைய கனெக்டாரின் பண்புகளை பின்வருமாறு அமைக்க வேண்டும்.

```
$ vim config/connect-file-source.properties
```

```
name=local-file-source
connector.class=FileStreamSource
tasks.max=1
file=/home/shrini/Desktop/smp.txt
topic=kaniyam
```

```
$ vim config/connect-file-sink.properties
```

```
name=local-file-sink
connector.class=FileStreamSink
tasks.max=1
file=sample.txt
topics=kaniyam
```

பின்னர் connect-standalone.sh எனும் பிராசஸை இயக்குவதன் மூலம் இச்செயல் முழுமையுற்று நம்முடைய தற்போதைய டைரக்டரியில் sample.txt இடம் பெற்றிருப்பதைக் காணலாம். டெர்மினலிலும் ஒருமுறை டாப்பிக்கை இயக்கி கோப்பில் உள்ள செய்திகள் இங்கு வெளிப்படுகின்றனவா என சோதித்துக் கொள்ளலாம்.

Terminal6:

```
$ bin/connect-standalone.sh config/connect-standalone.properties config/connect-file-source.properties config/connect-file-sink.properties
```

```
shrini@nithya-Lenovo-ideapad:~/kafka_2.12-2.5.0$ ls
bin  config  libs  LICENSE  logs  NOTICE  sample.txt  site-docs
shrini@nithya-Lenovo-ideapad:~/kafka_2.12-2.5.0$ cat sample.txt
It's a website for open-source technologies.
```

```
shrini@nithya-Lenovo-ideapad:~/kafka_2.12-2.5.0$ bin/kafka-console-consumer.sh --bootstrap-server local
{"schema":{"type":"string","optional":false},"payload":"அந்நியர்களிடமிருந்து விடுதலை பெற்றோம்."}
{"schema":{"type":"string","optional":false},"payload":"சொந்த நாட்டின் சில பிற்போக்கு மனிதர்களிடமிருந்தும் பிற
{"schema":{"type":"string","optional":false},"payload":"ஒரு தேசத்தில் உணவு, உடை, தொழில் சுதந்திரம் எல்லோ
{"schema":{"type":"string","optional":false},"payload":"இன்றின்றளவும் ஒரு தேசிய நற்போக்கு நிலையில் இத
{"schema":{"type":"string","optional":false},"payload":"இந்த சூழ்நிலையும் திறக்கிறது. "}
{"schema":{"type":"string","optional":false},"payload":"It's a website for open-source technologies."}
^CProcessed a total of 6 messages
```


9.6 Data Streaming: Pykafka

கஃப்காவின் புரொடியூசர் கன்ஸ்யூமர் ஆகியவற்றை பைத்தான் மொழியில் எழுதி செயல்படுத்த pykafka என்பது பயன்படுகிறது. 5 செகண்டுக்கு ஒருமுறை ஒரு json வடிவ தரவினை வெளிப்படுத்தும் producer.py எனும் புரோகிராம் இங்கு பின்வருமாறு எழுதப்பட்டுள்ளது.

https://gist.github.com/nithyadurai87/089138a14b0bca8024833fd3af03e54d

producer.py

```
from pykafka import KafkaClient
import json
import time

client = KafkaClient(hosts='localhost:9092')
topic = client.topics['dataets']
producer = topic.get_sync_producer()
producer.produce(b'test message')

for e in range(1000):
    data = {'number' : e}
    print(data)
    info_as_json = json.dumps(data)
    producer.produce(info_as_json.encode('ascii'))
    time.sleep(5)
```

இதனை இயக்கினால் அது பின்வருமாறு வெளிப்படுத்தும். நமக்குப் புரிய வேண்டும் என்பதற்காக இதனை பிரிண்ட் செய்து பார்த்துள்ளோம். ஆனால் உண்மையில் இது ஐந்து செகண்டுக்கு ஒருமுறை dataets எனும் டாப்பிக்குக்குள் இதனை செலுத்திக் கொண்டே இருக்கும்.

```
shrini@nithya-Lenovo-ideapad:~/kafka_2.12-2.5.0$ ls
bin  config  libs  LICENSE  logs  NOTICE  producer.py  sample.txt  site-docs
shrini@nithya-Lenovo-ideapad:~/kafka_2.12-2.5.0$ python3 producer.py
{'number': 0}
{'number': 1}
{'number': 2}
{'number': 3}
```

இந்த டாப்பிக்கை கன்சோலில் பிரிண்ட் செய்து பார்ப்பதன் மூலம் இதனை உறுதிபடுத்திக் கொள்ளலாம்.

```
$ bin/kafka-console-consumer.sh --bootstrap-server
localhost:9092 --topic dataets --from-beginning
```

```
shrini@nithya-Lenovo-ideapad:~/kafka_2.12-2.5.0$ bin/kafka-console-consumer.sh -
-bootstrap-server localhost:9092 --topic dataets --from-beginning
test message
{"number": 0}
{"number": 1}
{"number": 2}
{"number": 3}
{"number": 4}
```

இது ப்ரொடியூசருக்கான உதாரணம் மட்டுமே. கன்ஸ்யூமர் என்பது ஒரு டாப்பிக்குக்குள் இருப்பவற்றை வேறொரு இடத்தில் சேமிப்பது ஆகும். பொதுவாக mongodb என்பது கன்ஸ்யூமராக அமையும். இதற்கடுத்து இதனை mongodb-ல் சேமிப்பதற்கான கன்ஸ்யூமர் ப்ரோக்ராமை பைத்தான் மொழியில் எழுதுவது பற்றி பார்க்கலாம். அதற்கு முன்னர் mongodb-ஐப் பற்றிக் கொஞ்சம் தெரிந்து கொள்வோம்.

10. MongoDB

MongoDB என்பது திறந்த மூல மென்பொருள் கருவி ஆகும். இது NoSQL-ஐ அடிப்படையாகக் கொண்டு இயங்குகின்ற டேட்டாபேஸ் சேவையகம் ஆகும். அதாவது அட்டவணைகளில் சேமிக்க இயலாத தரவு அமைப்புகளையும் சேமிக்க வழிவகை செய்யும் டேட்டாபேசுக்கு NoSQL என்று பெயர். இதில் கீழ்க்கண்டவாறு பல்வேறு வகைகள் உள்ளன.

Document oriented - MongoDB,CouchDB

column oriented - Cassandra,Hbase

key value - Redis, Riak

graph – Neo4j,GraphDB

இவற்றுள் Mongo DB- ஐப் பற்றி இப்பகுதியில் காணலாம். இதுவும் key-value pair முறையில் அமையும் json வடிவ தரவுகளை சேமிக்கக் கூடியதுதான். ஆனால் சேமிக்கும்போது அதனை பைனரியாக மாற்றி bson (binary form of json) வடிவில் டாக்குமென்ட்டாக சேமிப்பதால் இது document oriented-க்கான டேட்டாபேஸாக அமைகிறது.

10.1 Installation

1. முதலில் MongoDB-க்கான பப்ளிக் GPG Key-ஐ (GNU Privacy Guard) அதற்கான முகவரியில் சென்று இறக்குமதி செய்து கொள்ளவும். இது ஓகே என்று வெளிப்படுத்தாமல், gnupg இல்லை என்று வெளிப்படுத்தினால், முதலில் அதனை நிறுவிவிட்டு பின்னர் இக்கட்டளையை அளிக்கவும்.

```
$ wget -qO - https://www.mongodb.org/static/pgp/server-4.2.asc | sudo apt-key add -
```

2. அடுத்து MongoDB-க்கான பேக்கேஜை இறக்குமதி செய்து கொள்ளவும். நமது உபுண்டு கணினியில் பயோனிக் & ஜெனியல் என்று இரண்டு பதிப்புகள் உள்ளன. lsb_release -a எனக் கொடுத்து என்ன பதிப்பு உள்ளது என்பதைத் தெரிந்து கொண்டு அதற்கேற்றவாறு கீழ்கண்ட கட்டளையை அளிக்கவும். xenial என வெளிப்படுத்தினால் கீழ்கண்ட கட்டளையில் ubuntu bionic என்ற இடத்தில் ubuntu xenial எனக் கொடுக்கவும்.

```
$ echo "deb [ arch=amd64 ] https://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/4.2 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-4.2.list
```

```
shrini@nithya-Lenovo-ideapad:~$ wget -qO - https://www.mongodb.org/static/pgp/server-4.2.asc | sudo apt-key add -
OK
shrini@nithya-Lenovo-ideapad:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.1 LTS
Release:        20.04
Codename:       focal
shrini@nithya-Lenovo-ideapad:~$ echo "deb [ arch=amd64 ] https://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/4.2 
db-org-4.2.list
deb [ arch=amd64 ] https://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/4.2 multiverse
```

3. Advanced Package Tool (APT) ரெப்பாசிட்டரியை புதுப்பிப்பதற்கான கட்டளையை இயக்கி விட்டு பின்னர் mongodb-ஐ நிறுவுவதற்கான கட்டளையை இயக்கவும்.

```
$ sudo apt-get update
$ sudo apt-get install -y mongodb-org
```

4. கீழ்க்கண்ட கட்டளைகள் , mongo db-ன் சேவைகளைத் துவங்குதல், நிறுத்துதல் நிலைப்பாட்டினை வெளிப்படுத்துதல் போன்ற வேலைகளைச் செய்கிறது. .

```
$ sudo service mongod start
$ sudo service mongod stop
$ sudo service mongod restart
$ sudo service mongod status
```

```
shrini@nithya-Lenovo-ideapad:~$ sudo service mongod start
shrini@nithya-Lenovo-ideapad:~$ sudo service mongod status
● mongod.service - MongoDB Database Server
     Loaded: loaded (/lib/systemd/system/mongod.service; disabl
     Active: active (running) since Fri 2020-08-28 09:49:44 IST
       Docs: https://docs.mongodb.org/manual
   Main PID: 15861 (mongod)
     Memory: 59.4M
     CGroup: /system.slice/mongod.service
             └─15861 /usr/bin/mongod --config /etc/mongod.conf

Aug 28 09:49:44 nithya-Lenovo-ideapad systemd[1]: Started Mongo
```

10.2 Mongo Shell

மாங்கோவுக்கான சேவையைத் துவங்கியவுடன் நமது டெர்மினலில் மாங்கோ எனக் கொடுத்து உள் நுழைந்தால் அதனுடன் வினவுவதற்குத் தேவையான இடத்தை நாம் சென்றடைந்து விடுவோம்.

```
$ mongo
```

```
shrini@nithya-Lenovo-ideapad:~$ mongo
MongoDB shell version v4.2.9
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongo
Implicit session: session { "id" : UUID("33096b97-c5bc-4ec0-8c18-7570f10cc551") }
MongoDB server version: 4.2.9
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        https://docs.mongodb.com/
Questions? Try the MongoDB Developer Community Forums
```

உள் நுழைந்தவுடன் என்னென்ன தரவுத்தளங்கள் உள்ளது எனக் காண,

```
> show dbs
```

பின் ஒரு புதிய தரவுத்தளத்தை உருவாக்க,

```
> use exams
```

டேட்டாபேசை உருவாக்கிய உடன் அதனைக் காண இயலாது. ஏதேனும் தரவுகளை உட்செலுத்தி show dbs எனக் கொடுத்தால்தான் அதனைக் காணலாம்.

```
> db.exams.insert({"Language":"Tamil,English"})
> show dbs
```

```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
> use exams
switched to db exams
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
> db.exams.insert({"Language":"Tamil,English"})
WriteResult({ "nInserted" : 1 })
```

```
> show dbs
admin   0.000GB
config  0.000GB
exams   0.000GB
local   0.000GB
```

அடுத்து exams எனும் டேட்டாபேசுக்குள் staff என்ற புதிய கலெக்சனை உருவாக்க மற்றும் தரவுகளை உட்செலுத்த,

```
> db.staff.insert({"name":"Perumal","age":42})
```

இங்கு db.staff என்பது exams.staff என்பதைக் குறிக்கிறது. இவ்வாறாக டேட்டாபேஸ் மற்றும் கலெக்சன் இரண்டையும் இணைத்துக் குறிப்பிடுவதே namespace ஆகும்.

இதுவரை உருவாக்கப்பட்ட கலெக்சன்களைக் காண,

```
> show collections
```

```
> db.staff.insert({"name":"Perumal","age":42})
WriteResult({ "nInserted" : 1 })
> show collections
exams
staff
```

இங்கு exams எனும் டேட்டாபேசில் இருக்கும் அனைத்து கலெக்சன்களையும் பட்டியலிடுகிறது.

ஒன்றுக்கும் மேற்பட்ட json கோப்புகளை ஒரு கலெக்சனில் சேர்க்க,

```
> db.books.insert([
{_id: ObjectId(),title: 'PHP in tamil',author:
'Priya',cond:'good',url:'http://freetamilebooks.com'},
{_id: ObjectId(),title: 'Mysql in Tamil',author: "Nithya D",
cond:'Average', url:'http://freetamilebooks.com',likes: 20,comments:
[{user:'Mahesh',message: 'Very useful madam', dateCreated: new
Date(2015,07,25,4,45),like: 1}]}
])
```

```
> db.books.insert({_id: ObjectId(),title: 'Sila nerangalil sila manithargal',author: 'Jayakanthan',
WriteResult({ "nInserted" : 1 })
> db.books.insert([
... {_id: ObjectId(),title: 'PHP in tamil',author: 'Priya',cond:'good',url:'http://freetamilebooks.c
... {_id: ObjectId(),title: 'Mysql in Tamil',author: "Nithya D", cond:'Average', url:'http://freetam
ful madam', dateCreated: new Date(2015,07,25,4,45),like: 1}]}
... ])
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 2,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
```

ஒரு கலெக்சனில் சேமிக்கப்பட்டுள்ளவற்றைக் காண,

```
>db.books.find()
```

```
> db.books.find()
{ "_id" : ObjectId("5f488d2acf5850a5379c4216"), "title" : "Sila neran
{ "_id" : ObjectId("5f488d57cf5850a5379c4217"), "title" : "Sila neran
{ "_id" : ObjectId("5f488f43cf5850a5379c4218"), "title" : "PHP in tam
{ "_id" : ObjectId("5f488f43cf5850a5379c4219"), "title" : "Mysql in T
0, "comments" : [ { "user" : "Mahesh", "message" : "Very useful madam
```

இதில் _id என்பதுதான் ஒவ்வொரு document-க்குமான object id ஆகும். இதுவே அதற்கான index-ஆக அமையும். ஒரு கலெக்சனை நீக்க db.staff.drop() என அதன் பெயரைக் குறிப்பிட்டுக் கூற வேண்டும். ஆனால் ஒரு டேட்டாபேஸை நீக்க dp.dropDatabase() எனக் கொடுத்தால் போதுமானது. தற்போது இயக்கத்தில் இருக்கும் டேட்டாபேஸ் நீங்கிவிடும். மேலும் டேட்டாபேசிலிருந்து document-ஐ நீக்குவது என்பது டிஸ்க்கிலிருந்தே அதனை நீக்குவதற்குச் சமம் ஆகும்.

10.3 Kafka to MongoDB

இப்போது கஃப்காவின் புரொடியூசர் கன்ஸ்யூமர் ஆகிய இரண்டையும் பைத்தான் மொழியில் எழுதுவோம். ஏற்கனவே புரொடியூசருக்கான ப்ரோக்ராம் எழுதப்பட்டு விட்ட நிலையில் இப்போது கன்ஸ்யூமராக மாங்கோ db-ஐ அமைப்பதற்கான புரோகிராம் கீழ்க்கண்டவாறு. இதனை consumer.py எனும் பெயரில் எழுதி சேமிக்கவும்.

https://gist.github.com/nithyadurai87/e2ac01eaeed67b1749b66c03ec167ec0

consumer.py

```
from pykafka import KafkaClient
from pymongo import MongoClient
import json
import sys

K_client = KafkaClient(hosts='localhost:9092')
topic = K_client.topics['dataets']
consumer =
topic.get_simple_consumer(consumer_timeout_ms=5000)

M_client = MongoClient('localhost',27017)
db = M_client.records
collection = db.data

counter = 1
for i in consumer:
   i = {str(counter):str(i.value.decode("utf-8")) }
   collection.insert_one(i)
   counter = counter + 1
```

ஏற்கனவே புரொடியூசருக்கான ப்ரோக்ராம் இயங்கி dataets என்கிற டாப்பிக்கிற்குள் தரவுகளை செலுத்திக் கொண்டே இருக்கும். இப்போது இந்த கன்ஸ்யூமருக்கான ப்ரோக்ராமை இயக்கும் போது அது dataets என்கிற டாப்பிக்கிலிருந்து எடுத்து மாங்கோ db-ல் தரவுகளை செலுத்திக் கொண்டே வரும்.

```
shrini@nithya-Lenovo-ideapad:~/kafka_2.12-2.5.0$ ls
bin  config  consumer.py  libs  LICENSE  logs  NOTICE  producer.py  sample.txt  site-docs
shrini@nithya-Lenovo-ideapad:~/kafka_2.12-2.5.0$ python3 consumer.py
shrini@nithya-Lenovo-ideapad:~/kafka_2.12-2.5.0$
```

ஆகவே இத்தகைய புரொடியூசர் மற்றும் கன்ஸ்யூமர் ஆகிய இரண்டையும் இணைக்கின்ற டாப்பிக் என்ற ஒன்றுதான் இங்கு channel என்று அழைக்கப்படும். இத்தகைய சேனல் வழியேதான் இவை இரண்டும் பேசிக்கொள்ளும். இப்போது மாங்கோவில் சென்று பார்த்தால் நிரலில் கொடுக்கப்பட்டுள்ளது போல records எனும் db

உருவாக்கப் பட்டிருப்பதைக் காணலாம். அதற்குள் data எனும் கலெக்சனில் தரவுகள் சேமிக்கப்பட்டுள்ளதை அறியலாம்.

```
> show dbs
admin    0.000GB
config   0.000GB
exams    0.000GB
local    0.000GB
records  0.000GB
test_db  0.000GB
> use records
switched to db records
> db.data.find()
{ "_id" : ObjectId("5f4f32bfa466abb288c1dac1"), "1" : "test me
{ "_id" : ObjectId("5f4f32bfa466abb288c1dac2"), "2" : "{\"numb
{ "_id" : ObjectId("5f4f32bfa466abb288c1dac3"), "3" : "{\"numb
{ "_id" : ObjectId("5f4f32bfa466abb288c1dac4"), "4" : "{\"numb
```

அடுத்ததாக டேட்டாபேஸில் இருக்கும் தரவுகளை backup எடுத்தல், மற்றொரு சர்வருக்கு மாற்றுதல் போன்ற பல்வேறு பணிகளைத் திட்டமிட்டு நடத்த உதவும் Airflow எனும் கருவியைப் பற்றிக் கொஞ்சம் பார்க்கலாம். at, crontab போன்ற கட்டளைகள் ஏற்கனவே லினக்ஸ்சில் இருந்தாலும் தற்போது பிக் டேட்டா என்று வரும்போது, பல்லாயிரக்கணக்கான கிளஸ்டர்களில் ஓடிக்கொண்டிருக்கும் பல்வேறு பணிகளை மேற்பார்வை செய்யும் பொறுப்பு நம்மிடம் வருகிறது. இதற்கான வசதி வெறும் அக்கட்டளைகளில் இல்லை. ஆகவே இதற்கென ஒரு UI interface தேவைப்படுகிறது. இந்தத் தேவையை பூர்த்தி செய்வதற்காக வந்ததே Airflow ஆகும். இதைப் பற்றி அடுத்த பகுதியில் காணலாம்.

11. Airflow

Airflow என்பது அப்பாச்சி நிறுவனம் வழங்குகின்ற திறந்த மூல மென்பொருள் கருவி ஆகும். கணினியில் நடைபெறும் ஒரு சில செயல்கள் தொடர்ச்சியாக எப்போதெல்லாம் நடைபெற வேண்டும் எனத் திட்டமிடுவது workflow scheduling எனப்படும். இவ்வாறு அதிக அளவில் திட்டமிடப்பட்ட பணிகளின் எண்ணிக்கை உயர உயர அவற்றைக் கண்காணிப்பது கடினமாகி விடுகிறது. இப்பிரச்சினைக்காக Airbnb என்ற நிறுவனம் முதன்முதலில் Airflow என்ற கருவியை உருவாக்கியது. இக்கருவி திட்டமிடப்பட்டு நடைபெற்றுக்கொண்டிருக்கும் பல்வேறு பணிகளை UI எனும் இடைமுகப்புத் திரை மூலம் மேலாண்மை செய்கிறது. ஒன்று, இரண்டு எனும் எண்ணிக்கையில் jobs ஓடிக் கொண்டிருக்கும்போது எந்தப் பிரச்சனையும் இல்லை. ஆனால் ஆயிரக்கணக்கில் அவற்றின் எண்ணிக்கை உயரும் போதும், பல்வேறு கணினிகளில் பல்வேறு கால அளவில் அவை ஓடிக் கொண்டிருக்கும் போதும், ஏதேனும் சிக்கல் ஏற்பட்டு ஒன்றிரண்டு jobs நின்றுவிட்டாலோ தோல்வியுற்றாலோ அவற்றைத் தெரிந்து கொள்வதற்கு இது வாய்ப்பளிக்கிறது. இதற்கென ஒரு இடைமுகப்புத் திரையை (UI Interface) Airflow வழங்குகிறது.

இதனைப் பற்றித் தெரிந்து கொள்வதற்கு முன்னர், முதலில் jobs-ஐ உருவாக்கி schedule செய்வது எப்படி என்று பார்ப்போம்.

11.1 Scheduling Jobs

இப்பகுதியில் தொடர்ச்சியான கால இடைவெளிகளில் MySQL டேட்டாபேசை backup எடுத்தல் மற்றும் அவற்றை ரிமோட் சர்வருக்கு அனுப்புதல் போன்ற இருவேறு பணிகளை உருவாக்கி Schedule செய்து பார்க்கப் போகிறோம்.

பணி 1:
ஒருமணி நேரத்திற்கு ஒருமுறை டேட்டாபேசை நமது கணினியில் backup எடுத்து சேமித்து வைத்துக் கொள்வது.

பணி 2:
இவ்வாறு சேமிக்கப்பட்ட கோப்பினை ஒரு நாளைக்கு ஒருமுறை வேறு ஒரு சர்வருக்கு அனுப்பி பாதுகாப்பாக வைத்துக்கொள்வது. இதில் உள்ள படிகள் பின்வருமாறு.

படி 1:
முதலில் mysql டேட்டாபேசுக்கான சேவையைத் துவக்கவும். பின் பயனர் பெயர் மற்றும் கடவுச்சொல் கொடுத்து உள்நுழையவும்.

```
$ sudo service mysql start
$ mysql -u root -p
```

```
shrini@nithya-Lenovo-ideapad:~$ sudo service mysql start
shrini@nithya-Lenovo-ideapad:~$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.22-0ubuntu0.20.04.2 (Ubuntu)

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All
```

படி 2:
என்னென்ன டேட்டாபேஸ் உள்ளது என கீழ்க்கண்ட கட்டளை மூலம் தெரிந்து கொள்ளவும். இதில் sample எனும் பெயரில் உள்ள டேட்டாபேசைத்தான் ஒருமணி நேரத்திற்கு ஒருமுறை பேக்கப் எடுக்கப் போகிறோம்.

```
mysql> show databases;
```

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| metastore          |
| mysql              |
| performance_schema |
| sample             |
| sys                |
| world              |
| wsexport           |
+--------------------+
8 rows in set (0.00 sec)
```

படி 3:
அதற்கு முன்னர் அதில் என்னென்ன அட்டவணைகள் மற்றும் தரவுகள் உள்ளன எனக் கீழ்க்கண்ட கட்டளைகள் மூலம் தெரிந்து கொள்ளவும்.

```
mysql> use sample;
mysql> show tables;
```

```
mysql> use sample;
Reading table information for co
You can turn off this feature to

Database changed
mysql> show tables;
+------------------+
| Tables_in_sample |
+------------------+
| info             |
+------------------+
1 row in set (0.01 sec)

mysql> select * from info;
+------+--------+------+
| id   | name   | age  |
+------+--------+------+
|    1 | Aarthi |   25 |
|    2 | Sathya |   18 |
+------+--------+------+
2 rows in set (0.03 sec)
```

படி 4:
பின் ctrl-d எனக் கொடுத்து mysql-க்கான ஷெல்லில் இருந்து வெளிவந்த பின்னர்
mysqldump எனும் கட்டளை மூலம் நாம் விரும்பும் டேட்டாபேசை பிரதி எடுத்து வைத்துக் கொள்ளலாம். இங்கு இக்கட்டளையைத் தொடர்ந்து root என்பது பயனர் பெயராகவும், password என்பது கடவுச்சொல்லாகவும் கொடுக்கப்பட்டுள்ளது. பின் sample எனும் டேட்டாபேஸ் sample.sql எனும் பெயரில் நமது கணினியின் தற்போதைய டைரக்டரியில் ஒரு கோப்பாக சேமிக்கப்படுகிறது..

```
$ mysqldump -u root -ppassword sample > sample.sql
```

```
mysql> ^DBye
shrini@nithya-Lenovo-ideapad:~$ mysqldump -u root -ppassword sample > sample.sql
mysqldump: [Warning] Using a password on the command line interface can be insecure.
shrini@nithya-Lenovo-ideapad:~$
```

இக்கோப்பினை எங்கு வேண்டுமானாலும் அனுப்பி mysql-க்குள் பின்வருமாறு இம்போட் செய்து கொள்ளலாம்.

```
mysql -u root -ppassword sample < sample.sql;
```

ஆகவே job2 என்பது இக்கோப்பினை ஒரு ரிமோட் சர்வருக்கு அனுப்புவதற்கான கட்டளைகளை உள்ளடக்கியிருக்கும். அதற்கு முன்னர் job1- ஐ உருவாக்கி schedule செய்வோம்.

படி 5:
job1 என்பது டேட்டாபேசை பேக்கப் எடுப்பதற்கான கட்டளையை ஒரு ஷெல் ஸ்கிரிப்டிலிட்டு, பின் அந்த ஸ்கிரிப்ட் ஒருமணி நேரத்திற்கு ஒருமுறை ஓட வேண்டும் என crontab கொண்டு அமைப்பதன் மூலம் உருவாக்கப்படும். இது பின்வருமாறு..

```
$ vim bkp.sh
mysqldump -u root -ppassword sample >
"/home/shrini/backups/"sample_"$(date +"%Y%m%d_%H%M%S").sql"
```

```
shrini@nithya-Lenovo-ideapad:~$ vim bkp.sh
shrini@nithya-Lenovo-ideapad:~$ cat bkp.sh
mysqldump -u root -ppassword sample > "/home/shrini/backups/"sample_"$(date +"%Y%m%d_%H%M%S").sql"
```

இந்த ஸ்கிரிப்டை பின்வருமாறு இயக்கி கோப்பு சேமிக்கப்படுகிறதா என சோதித்துக் கொள்ளவும்.

```
$ sh bkp.sh
$ ls /home/shrini/backups/
```

```
shrini@nithya-Lenovo-ideapad:~$ sh bkp.sh
mysqldump: [Warning] Using a password on the command line interf
ure.
shrini@nithya-Lenovo-ideapad:~$ ls /home/shrini/backups/
sample_20201104_190848.sql
```

எந்த இடத்தில் கோப்பினை சேமிக்க வேண்டும் என்பதும், எந்தப் பெயரில் சேமிக்க வேண்டும் என்பதும் ஸ்கிரிப்ட்க்குள் கொடுக்கப்பட்டுள்ளது. வெறும் sample.sql எனும் பெயரில் சேமிக்காமல் ஒவ்வொரு முறையும் அப்போதைய நேரத்துடன் சேர்த்து சேமிக்கிறோம். ஏனெனில் ஒரே பெயரில் அடுத்தடுத்து கோப்புகள் சேமிக்கப்படுவதை இது தவிர்க்கிறது.

படி 6:
கோப்புகள் சேமிக்கப்பட்டுள்ள இடத்திலிருந்து வேறு ஒரு இடத்திற்கு அதனை அனுப்பும் வேலையை rsync எனும் கட்டளை செய்கிறது. இக்கட்டளையைத் தொடர்ந்து நமது கணினியில் கோப்புகள் உள்ள இடத்தின் பாதையைக் கொடுக்க வேண்டும். பின் அதைத்தொடர்ந்து ரிமோட் சர்வரின் ஐபி முகவரி, பயனர் பெயர் மற்றும் அக்கணினியில் எந்த இடத்தில் சேர்க்க வேண்டும் ஆகிய விவரங்களைப் பின்வருமாறு அளிக்க வேண்டும்.

```
$ rsync -arv /home/shrini/backups shrini@139.59.47.5:/home/shrini
```

```
shrini@nithya-Lenovo-ideapad:~$ rsync -arv /home/shrini/backups shrini@139.59.47
.5:/home/shrini
shrini@139.59.47.5's password:
sending incremental file list
backups/
backups/sample_20201104_190848.sql

sent 2,046 bytes  received 39 bytes  245.29 bytes/sec
total size is 1,885  speedup is 0.90
shrini@nithya-Lenovo-ideapad:~$
```

-arv என்பது archive, recursive, verbose எனப் பொருள்படும். இதற்கும் FTP-க்கும் ஒரு சிறு வித்தியாசம் உள்ளது. இரண்டையுமே முதல் முறை

இயக்கும்போது ஓரிடத்திலிருந்து அடுத்த இடத்திற்கு கோப்புகளை கொண்டு சேர்க்கும் வேலையைத் தான் செய்கிறது. ஆனால் அதையே மறுமுறை இயக்கும்போது rsync என்பது கோப்புகள் சேர்க்கப்பட வேண்டிய இடத்தில் அவை ஏற்கனவே இருப்பின் அவற்றை விடுத்து, புதிய கோப்புகளை மட்டும் அனுப்பும். மேலும் ஏற்கனவே இருக்கும் கோப்புகளில் ஏதேனும் மாற்றங்கள் இருந்தால் அக்கோப்பினையும் அனுப்பும். எவையெல்லாம் மாற்றம் செய்யப்பட்டவை எனக் கண்டுபிடிக்க md5sum எனும் கட்டளையை rsync பயன்படுத்துகிறது. இக்கட்டளையானது ஒவ்வொரு கோப்பின் மீதும் செயல்பட்டு அதற்கென்று ஒரு நெடிய string-ஐ உருவாக்கும். அதுவே அக்கோப்பிற்கான அடையாளமாக அமையும். இரண்டு இடங்களிலும் இக்கட்டளை மூலம் உருவாக்கப்படும் நெடிய string ஒத்துப் போகவில்லை என்றால் கோப்புகளில் ஏதோ மாற்றம் நிகழ்ந்துள்ளது என்று அர்த்தம். ஆகவே அதனை மட்டும் rsync அனுப்பும். ஆனால் FTP என்பது ஒவ்வொரு முறையும் அனைத்துக் கோப்புகளையும் மொத்தமாகக் கொண்டு சேர்க்கும். ஆகவே ஏற்கனவே அவை இருந்தால் கூட அதன் மீதே overwrite செய்யும்.

படி 7:
job2 என்பது இந்த rsync கட்டளையை ஒரு ஷெல் ஸ்கிரிப்டிலிட்டு, பின் அந்த ஸ்கிரிப்ட் ஒரு நாளைக்கு ஒருமுறை ஓட வேண்டும் என crontab கொண்டு அமைப்பதன் மூலம் உருவாக்கப்படும். ஆனால் மேற்கண்ட படியில் rsync-ஐ இயக்கும்போது ஒவ்வொரு முறையும் அது நம்மிடம் பாஸ்வேர்ட் கேட்டு நிற்பதைக் காணலாம். நாம் பாஸ்வேர்ட் வழங்கிய பின்னரே தனது செயல்பாட்டைத் தொடங்கும்.

ஒரு ஷெல் ஸ்கிரிப்ட் என்பது தன்னிச்சையாக இயங்கி முடியக்கூடிய கட்டளைகளைத் தான் பெற்றிருக்க வேண்டும். இதுபோன்று பாஸ்வேர்ட் கேட்டு நிற்கக்கூடாது. ஆகவே நாம் கோப்புகளை செலுத்த இருக்கும் கணினியின் பாஸ்வேர்டுக்கு இணையான அதன் pem file எங்கு இருக்கிறது என இக்கட்டளையுடன் சேர்த்துக் கொடுக்க வேண்டும். இதைத்தான் ஷெல் ஸ்கிரிப்ட் உள்ளும் கொடுக்க வேண்டும்.

```
$ rsync -arv -e "ssh -i /home/shrini/shrini-freepem.pem" /home/shrini/backups ubuntu@35.166.185.40:/home/ubuntu
```

```
shrini@nithya-Lenovo-ideapad:~$ rsync -arv -e "ssh -i /home/shrini/shrini-freepe
m.pem" /home/shrini/backups ubuntu@35.166.185.40:/home/ubuntu
sending incremental file list
backups/
backups/sample_20201104_190848.sql

sent 2,046 bytes  received 39 bytes  278.00 bytes/sec
total size is 1,885  speedup is 0.90
shrini@nithya-Lenovo-ideapad:~$
```

ஒருமுறை pem file துணைகொண்டு பாஸ்வேர்ட் இல்லாமல் உள்நுழைய முடிகிறதா என ssh செய்து பார்த்துக் கொள்ளவும்.

```
$ ssh -i /home/shrini/shrini-freepem.pem ubuntu@35.166.185.40
```

```
shrini@nithya-Lenovo-ideapad:~$ ssh -i /home/shrini/shrini-freepem.pem ubuntu@35
.166.185.40
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 3.13.0-92-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

  System information as of Wed Nov  4 13:58:47 UTC 2020

  System load:  0.0               Processes:           122
```

```
(base) ubuntu@ip-172-31-29-250:~$ ls ./backups/
sample_20201104_190848.sql
(base) ubuntu@ip-172-31-29-250:~$ logout
Connection to 35.166.185.40 closed.
shrini@nithya-Lenovo-ideapad:~$
```

பின் ஷெல் ஸ்கிரிப்டை உருவாக்கவும். இதில் rsync முடிந்தவுடன் கோப்புகள் அனைத்தையும் நீக்கும் கட்டளை இணைக்கப்பட்டுள்ளது.

```
$ vim push_remote.sh

rsync -arv -e "ssh -i /home/shrini/shrini-freepem.pem"
/home/shrini/backups ubuntu@35.166.185.40:/home/ubuntu
rm -f /home/shrini/backups/*
```

```
shrini@nithya-Lenovo-ideapad:~$ vim push_remote.sh
shrini@nithya-Lenovo-ideapad:~$ cat push_remote.sh
rsync -arv -e "ssh -i /home/shrini/shrini-freepem.pem" /home/shrini/backups ubuntu@35.166.185.40
rm -f /home/shrini/backups/*
shrini@nithya-Lenovo-ideapad:~$ sh push_remote.sh
sending incremental file list

sent 115 bytes  received 17 bytes  264.00 bytes/sec
total size is 1,885  speedup is 14.28
shrini@nithya-Lenovo-ideapad:~$
```

படி 8:
இரண்டு ஷெல் ஸ்கிரிப்டையும் இயக்குவதற்கு அனைத்து பயனர்களுக்கும் execute அனுமதி அளிக்கவும்.

```
$ chmod a+x bkp.sh
$ chmod a+x push_remote.sh
```

```
shrini@nithya-Lenovo-ideapad:~$ chmod a+x bkp.sh
shrini@nithya-Lenovo-ideapad:~$ chmod a+x push_remote.sh
shrini@nithya-Lenovo-ideapad:~$ ls -l *.sh
-rwxrwxr-x 1 shrini shrini   99 Nov  4 19:06 bkp.sh
-rw-rw-r-- 1 shrini shrini 1496 Mar 24  2018 install-nightly.sh
-rwxrwxr-x 1 shrini shrini  126 Nov  4 19:52 push_remote.sh
shrini@nithya-Lenovo-ideapad:~$
```

படி 9:
கடைசியாக crontab மூலம் bkp.sh எனும் ஸ்கிரிப்ட் ஒருமணி நேரத்திற்கு ஒருமுறை ஓடுமாறும், push_remote.sh எனும் ஸ்கிரிப்ட் ஒரு நாளைக்கு ஒருமுறை நள்ளிரவு 12:20 மணிக்கு ஓடுமாறும் பின்வருமாறு அமைக்கப்படுகிறது.

```
$ crontab -e
0 * * * * /bin/bash /home/shrini/bkp.sh
20 0 * * * /bin/bash /home/shrini/push_remote.sh
```

crontab -e எனக் கொடுத்து edit செய்த பின்னர், crontab -l எனக் கொடுத்து schedule செய்யப்பட்டுள்ள கட்டளைகளை சரிபார்த்துக் கொள்ளவும்.

```
$ crontab -l
```

```
shrini@nithya-Lenovo-ideapad:~$ crontab -e
crontab: installing new crontab
shrini@nithya-Lenovo-ideapad:~$ crontab -l
# Edit this file to introduce tasks to be run by cro
#
# Each task to run has to be defined through a singl
# indicating with different fields when the task wil
# and what command to run for the task
#
# To define the time you can provide concrete values
# minute (m), hour (h), day of month (dom), month (m
# and day of week (dow) or use '*' in these fields (
#
# Notice that tasks will be started based on the cro
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is s
# email to the user the crontab file belongs to (unl
#
# For example, you can run a backup of all your user
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
0 * * * * /bin/bash /home/shrini/bkp.sh
20 0 * * * /bin/bash /home/shrini/push_remote.sh
```

11.2 Schedule using Airflow

இனிவரும் பகுதியில் மேற்கண்ட அதே விஷயத்தை Airflow கொண்டு உருவாக்கித் திட்டமிடுவது பற்றிக் காணலாம். கீழ்கண்ட கட்டளைகள் Airflow இயங்குவதற்குத் தேவையான விஷயத்தை இன்ஸ்டால் செய்யும்..

```
$ sudo pip3 install apache-airflow
$ sudo pip3 install flask
$ sudo pip3 install flask_bcrypt
$ sudo pip3 install kombu==4.5.0
```

இவை இயங்கி முடிந்தவுடன் நமது கணினியின் ஹோம் டைரக்டரியில் airflow எனும் டைரக்டரி உருவாக்கப்படும். மேற்கண்ட பகுதியில் நாம் உருவாக்கிய இரண்டு பணிகளுக்கும் ஷெல் ஸ்கிரிப்ட்க்கு பதிலாக, இரண்டு பைத்தான் புரோகிராமை உருவாக்கி அவற்றை task1, task2 எனும் பெயரில் dags டைரக்டரியின் கீழ் (~/airflow/dags) சேமிக்கவும்.

https://gist.github.com/nithyadurai87/8104b9c364f9bd7c6a7b5aeafee24a79

task1:

```
from datetime import datetime
from airflow import DAG
import os
from airflow.operators.python_operator import PythonOperator

def backup():
```

```
    backup_cmd = 'mysqldump -u root -ppassword sample > "/home/shrini/backups/"sample_"$(date +"%Y%m%d_%H%M%S").sql"'
    os.system(backup_cmd)

dag = DAG('mysql_backup', description='Mysql backup every hour', schedule_interval='00 * * * *', start_date=datetime(2020, 11, 2), catchup=False)

backup_operator = PythonOperator(task_id='backup_task', python_callable=backup, dag=dag)
```

நிரலுக்கான விளக்கம்:

Airflow-வில் பல ஆப்பரேட்டர்கள் உள்ளன. அவற்றில் எதன் துணைகொண்டு நாம் இந்தச் செயலை செய்யப் போகிறோமோ அதனை முதலில் import செய்து கொள்ள வேண்டும். இங்கு பைத்தான் ஆப்பரேட்டரை இம்போட் செய்துள்ளோம். பின்பு backup எடுப்பதற்கான கட்டளையை ஒரு function-க்குள் எழுதி இயக்கியுள்ளோம். எப்போதும் ஒரு ஷெல் கட்டளையை பைத்தான் மொழியில் இயக்குவதற்கு os.system() எனும் method பயன்படும். பின்பு இதற்கான dag வரையறுக்கப்படுகிறது. அது தனக்கென ஒரு id, வரையறை மற்றும் எப்போதெல்லாம் இயங்க வேண்டும் ஆகிய மதிப்புகளைப் பெற்றிருப்பதைக் காணலாம். இந்த id தான் Airflow UI திரையில் வெளிப்படும். மேலும் start_date என்பது எந்தத் தேதியிலிருந்து இச்செயலை செய்ய வேண்டும் என்பதைக் குறிக்கிறது. அடுத்து catchup=False என்றிருந்தால், இடையில் சர்வரின் இயக்கம் தடைபடுகின்ற நேரத்தில், அது இயங்க வேண்டிய சுழற்சிகளை நிறுத்திவிடும். அதாவது Airflow server-ன் இயக்கம் இன்று நிறுத்தப்பட்டு நாளை மீண்டும் தொடக்கினால், தொடங்கிய நேரம் முதல் jobs ஓடத் தொடங்கும். அதுவே catchup=True என்றிருந்தால், விட்ட நேரத்திற்கும் சேர்த்து இயங்கி முடிக்கும்.

கடைசியாக இந்த dag மற்றும் function ஆகியவற்றை மதிப்புகளாகக் கொண்ட ஒரு பைத்தான் ஆப்பரேட்டர் இச்செயலுக்காக உருவாக்கப்படுகிறது.

task2:
https://gist.github.com/nithyadurai87/55fa4ed6a2866ddcf53cea1958bcf1ed

```
from datetime import datetime
from airflow import DAG
import os
from airflow.operators.python_operator import PythonOperator

def pushfile():
    push_cmd = 'rsync -arv -e "ssh -i /home/shrini/shrini-freepem.pem" /home/shrini/backups ubuntu@35.166.185.40:/home/ubuntu'
    os.system(push_cmd)

def clr():
    clr_cmd = 'rm -f /home/shrini/backups/*'
    os.system(clr_cmd)

dag = DAG('push_remote', description='Pushing files to remote every day', schedule_interval='20 00 * * *', start_date=datetime(2020, 11, 2), catchup=False)

pushfile_operator = PythonOperator(task_id='pushfile_task', python_callable=pushfile, dag=dag)

clear_operator = PythonOperator(task_id='clearing_task', python_callable=clr, dag=dag)
pushfile_operator >> clear_operator
```

நிரலுக்கான விளக்கம்:

இங்கு backup எடுக்கப்பட்ட கோப்புகளை ரிமோட் சர்வருக்கு அனுப்புதல், அனுப்பிய பின் அவற்றை தற்போதைய டைரக்டரியில் இருந்து நீக்குதல் ஆகிய இரண்டு செயல்களுக்கும் இரண்டு தனித்தனி function எழுதப்பட்டுள்ளது. இச்செயல்கள் ஒவ்வொரு நாளும் நள்ளிரவு 12:20 மணிக்கு நடக்குமாறு schedule செய்யப்பட்டுள்ளது. அதன் பிறகு இவ்விரண்டு செயல்களுக்கும் தனித்தனியே இரண்டு ஆப்பரேட்டர்கள் உருவாக்கப்பட்டுள்ளன. முதல் ஆப்பரேட்டரின் செயல்பாடு முடிந்தவுடன் இரண்டாவது ஆப்பரேட்டரின் செயல்பாடு துவங்க வேண்டும் என்பதை >> எனும் குறியீடு குறிக்கிறது.

மேற்கண்ட இரண்டு பணிகளுக்கான நிரல்களையும் dags டைரக்டரியின் கீழ் (~/airflow/dags) சேமித்தவுடன் Airflow-வைத் துவக்குவதற்கான கட்டளையை அளிக்கவும்.

```
$ airflow initdb
```

இது 'Failed to import' என்பது போன்று ஏதேனும் error-ஐ வெளிப்படுத்தினால், நம்முடைய நிரல்களில் ஏதோ தவறு இருக்கிறது என்று அர்த்தம். ஆகவே இக்கட்டளை எவ்வித இடையூறும் இல்லாமல் ஓடி முடிந்தபின், இதற்கான webserver மற்றும் scheduler ஆகிய இரண்டையும் இரண்டு தனித்தனி டெர்மினல்களில் ஓட விடவும்.

```
$ airflow webserver
$ airflow scheduler
```

```
shrini@nithya-Lenovo-ideapad:~$ airflow initdb
DB: sqlite:////home/shrini/airflow/airflow.db
[2020-11-05 14:56:50,422] {db.py:378} INFO - Creating tables
INFO  [alembic.runtime.migration] Context impl SQLiteImpl.
INFO  [alembic.runtime.migration] Will assume non-transactional DDL.
/usr/local/lib/python3.8/dist-packages/airflow/models/dag.py:1337: PendingDep
id create_tag_template_field_result is already in the DAG. Starting in Airflo
  warnings.warn(
Done.
shrini@nithya-Lenovo-ideapad:~$ airflow webserver
  ____________       _____________
 ____    |__( )_________  __/__  /________      __
____  /| |_  /__  ___/_  /_ __  /_  __ \_ | /| / /
___  ___ |  / _  /   _  __/ _  / / /_/ /_ |/ |/ /
 _/_/  |_/_/  /_/    /_/    /_/  \____/____/|__/
[2020-11-05 14:57:55,464] {__init__.py:50} INFO - Using executor SequentialEx
[2020-11-05 14:57:55,470] {dagbag.py:417} INFO - Filling up the DagBag from /
/usr/local/lib/python3.8/dist-packages/airflow/models/dag.py:1337: PendingDep
id create_tag_template_field_result is already in the DAG. Starting in Airflo
  warnings.warn(
Running the Gunicorn Server with:
Workers: 4 sync
```

```
shrini@nithya-Lenovo-ideapad:~$ airflow scheduler
  ____________       _____________
 ____    |__( )_________  __/__  /________      __
____  /| |_  /__  ___/_  /_ __  /_  __ \_ | /| / /
___  ___ |  / _  /   _  __/ _  / / /_/ /_ |/ |/ /
 _/_/  |_/_/  /_/    /_/    /_/  \____/____/|__/
[2020-11-05 15:00:23,046] {__init__.py:50} INFO - Using executor Seq
tor
[2020-11-05 15:00:23,859] {scheduler_job.py:1367} INFO - Starting th
[2020-11-05 15:00:23,860] {scheduler_job.py:1375} INFO - Running exe
r -1 seconds
[2020-11-05 15:00:23,860] {scheduler_job.py:1376} INFO - Processing
 most -1 times
[2020-11-05 15:00:23,860] {scheduler_job.py:1379} INFO - Searching f
/home/shrini/airflow/dags
[2020-11-05 15:00:23,874] {scheduler_job.py:1381} INFO - There are 2
home/shrini/airflow/dags
[2020-11-05 15:00:23,874] {scheduler_job.py:1438} INFO - Resetting o
```

11.3 Airflow UI

webserver மற்றும் scheduler ஆகிய இரண்டும் தனித்தனி டெர்மினல்களில் ஓடிக்கொண்டிருக்கும் போது கீழ்க்கண்ட முகவரியில் சென்று காணவும். இதுதான் நாம் உருவாக்கி ஷெட்யூல் செய்துள்ள பணிகளைத் திரையில் காண உதவும் இடைமுகப்பு ஆகும்.

http://192.168.1.15:8080/admin/

Airflow - DAGs

Not secure | 192.168.1.15:8080/admin/?showPaused=True

Airflow | DAGs | Data Profiling | Browse | Admin | Docs | About | 2020-11-05 09:39:42 UTC

DAGs

Search:

		DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
	Off	example_bash_operator	0 0 * * *	Airflow	6	2020-11-03 05:14	19	
	Off	example_branch_dop_operator_v3	*/1 * * * *	Airflow				
	Off	example_branch_operator	@daily	Airflow				
	Off	example_complex	None	airflow				
	Off	example_external_task_marker_child	None	airflow				
	Off	example_external_task_marker_parent	None	airflow				
	Off	example_http_operator	1 day, 0:00:00	Airflow				

நம்முடைய dag பைத்தான் ஆப்பரேட்டர் கொண்டு எழுதப்பட்டது. இதேபோல bash ஆப்பரேட்டர், http ஆப்பரேட்டர், docker, hive, kubernetes என பலப்பல ஆப்பரேட்டர்கள் உள்ளன. அவை ஒவ்வொன்றுக்குமான உதாரணங்கள் தான் இங்கே இருப்பவை எல்லாம். இவற்றில் நம்முடைய dag-ஐ ஃபில்டர் செய்து கொள்ளவும்.

DAGs

Search:

		DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
	On	mysql_backup	00****	airflow		2020-11-05 08:00		
	On	push_remote	20 00***	airflow		2020-11-04 00:20		

Showing 1 to 2 of 2 entri

Show Paused DAGs

இதில் நம்முடைய dag-ன் பெயர், எப்போது schedule செய்யப்பட்டுள்ளது, சமீபமாக ஓடி முடிந்த dag-ன் நிலைகள், கடைசியாக எப்போது ஓடியது ஆகியவற்றைக் குறிக்கும் வகையில் பல்வேறு உப தலைப்புகள் உள்ளன. இவற்றில் Links எனும் உபதலைப்பின் கீழ் dag-ஐத் தூண்டுதல், அவற்றின் செயல்பாடுகளை வரைபட வடிவில் வெளிப்படுத்துதல், அவற்றுக்கான logs- ஐ வெளிப்படுத்துதல், மூல நிரலை வெளிப்படுத்துதல் போன்ற பல்வேறு செயல்களுக்கான ஐகான்கள் உள்ளன.

Trigger Dag Links

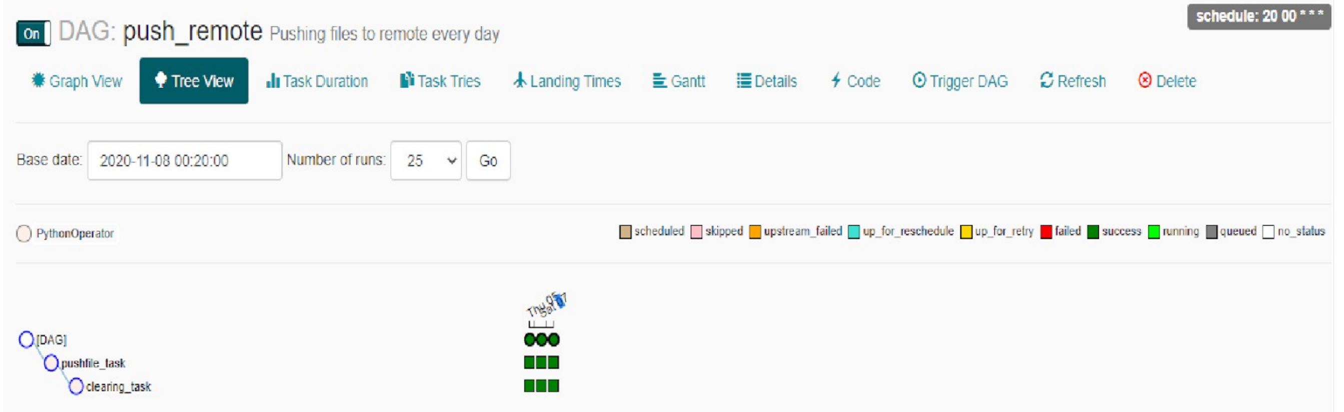
Tree View Links

Links Logs

இவற்றில் Trigger Dag எனும் ஐகானின் மீது சொடுக்கினால், DAG Runs எனும் உபதலைப்பின் கீழ் இளம்பச்சை நிறத்தில் ஒரு வட்டம் உருவாவதைக் காணலாம். அதாவது dag ஓடத் தொடங்கியிருக்கிறது என்று அர்த்தம். Schedule-ன் படி இல்லாமால், நமக்கு எப்போது தேவையோ அப்போது dag-ஐ ஓட்டிப் பார்க்க இந்த ஐகான் பயன்படும். Recent Tasks, DAG Runs ஆகிய உப தலைப்புகளின் கீழ் உள்ள பல்வேறு நிற வட்டங்கள் dag ஓட்டத்தின் பல்வேறு நிலைகளைக் குறிக்கின்றன. அவை கரும்பச்சையாக இருந்தால் வெற்றிகரமாக ஓடியது என்றும்,

இளம்பச்சையாக இருந்தால் ஓடிக்கொண்டிருக்கிறது என்றும், சிகப்பாக இருந்தால் ஓடித் தோல்வியுற்றது என்றும் அர்த்தம்.

அடுத்து Tree View ஐகானின் மீது சொடுக்கினால் அது பின்வருமாறு ஒரு வரைபடத்தை வெளிப்படுத்தும். அதேபோல் Graph view-மீதும் சொடுக்கிப் பார்க்கவும்.



Logs ஐகானின் மீது சொடுக்கினால் அது பின்வருமாறு வெளிப்படுத்தும். அதேபோல் மூல நிரலைக் காட்டும் ஐகானின் மீது சொடுக்கினால் அதற்கான நிரல் திரையில் வெளிப்படுவதைக் காணலாம்..

Logs

List (36) Add Filter

× Dag Id equals mysql_backup Reset Filters

Id	Dttm	Dag Id	Task Id	Event	Execution Date	Owner	Extra
242	11-09T04:58:26.368983+00:00	mysql_backup		tree		anonymous	[('dag_id', 'mysql_backup'), ('num_runs', '')]
237	11-09T04:55:17.361223+00:00	mysql_backup	backup_task	success	11-09T03:00:00+00:00	airflow	
236	11-09T04:55:17.053558+00:00	mysql_backup	backup_task	cli_run	11-09T03:00:00+00:00	shrini	{"host_name": "nithya-Lenovo-ideapad", "full_command": "['/usr/local/bin/airflow', 'run', 'mysql_backup', 'backup_task', '2020-11-09T03:00:00+00:00', '--local', '--pool', 'default_pool', '-sd', '/home/shrini/airflow/dags/task1.py']"}
235	11-09T04:55:16.939199+00:00	mysql_backup	backup_task	running	11-09T03:00:00+00:00	airflow	

11.4 Variables

backup எடுக்கும் பணிக்காக எழுதப்பட்ட பைத்தான் நிரலில் அக்கட்டளையைக் கொஞ்சம் கவனிக்கவும். Mysql-க்குள் உள் நுழைவதற்காக -u வைத் தொடர்ந்து பயனர்பெயர்(root) மற்றும் -p ஐத் தொடர்ந்து கடவுச்சொல் (password) ஆகியவை நேரடியாகவே கொடுக்கப்பட்டுள்ளன. ஒவ்வொரு முறை டேட்டாபேஸின் பாஸ்வேர்டை மாற்றும் போதும், புதிய பயனராக உள் நுழைய முயலும் போதும் இந்த மூல நிரலில் சென்று மாற்றம் செய்ய வேண்டும். இதற்கு பதிலாக இதற்கான variable-களை UI திரையில் உருவாக்கிக் கொண்டு அதனை நாம் நம்முடைய நிரல்களில் பயன்படுத்தலாம். இதன் மூலம் மதிப்பினை மாற்றினாலும் மூல நிரலில் கை வைக்கத் தேவையில்லை. UI திரையில் சென்று அந்த variable-களுக்கான மதிப்பினை மாற்றினால் போதும்.

முதலில் Airflow UI -ல் சென்று Action -> Variable -> Create ஆகிய இணைப்பின் மீது சொடுக்கவும். அது key, val ஆகிய பெட்டிகளைக் கொண்ட திரையை வெளிப்படுத்தும். அதில் நாம் உருவாக்க வேண்டிய variable-ன் பெயர் மற்றும் அதன் மதிப்பினை கீழ்கண்டவாறு அளிக்கவும்.

பின் list எனும் இணைப்பின் மீது சொடுக்கினால் நாம் உருவாக்கிய variable-களும் அவற்றின் மதிப்புகளும் பின்வருமாறு வெளிப்படும்.

Variables

Choose File No file chosen | Import Variables

List (3) | Create | Add Filter▾ | With selected▾ | Search: key, val, is_encrypted

		Key	Val	Is Encrypted
☐		mysql_password	********	⊙
☐		mysql_username	root	⊙
☐		name	shrini	⊙

திரையில் உருவாக்கப்பட்ட mysql_username, mysql_password ஆகியவை நம்முடைய நிரலுக்குள் import செய்யப்படுகின்றன. பின் இவ்விரு variable-களின் மதிப்பும் usr, pwd ஆகிய பெயரில் நிரலுக்குள் செலுத்தப்படுகின்றன. இது பின்வருமாறு.

https://gist.github.com/nithyadurai87/6c3a7035a4ce48372a6d8864e5f65a0b

```
from datetime import datetime
from airflow import DAG
import os
from airflow.operators.python_operator import PythonOperator
from airflow.models import Variable
usr = Variable.get("mysql_username")
pwd = Variable.get("mysql_password")

def backup():
```

```
    backup_cmd = 'mysqldump -u '+usr+' -p'+pwd+' sample > 
"/home/shrini/backups/"sample_"$(date +"%Y%m%d_%H%M
%S").sql"'
    os.system(backup_cmd)

dag = DAG('mysql_backup', description='Mysql backup every 
hour', schedule_interval='00 * * * *', 
start_date=datetime(2020, 11, 2), catchup=False)

backup_operator = PythonOperator(task_id='backup_task', 
python_callable=backup, dag=dag)
```

11.5 Bash Operator

மேற்கண்ட அதே நிரலை பைத்தான் ஆப்பரேட்டர் இல்லாமல் bash ஆப்பரேட்டர் கொண்டு உருவாக்கப்பட்ட ப்ரோக்ராம் பின்வருமாறு.

https://gist.github.com/nithyadurai87/af1d2e82148bb56041d80e2495080bc7

```
from datetime import datetime
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
```

```
dag = DAG('bash_exp', description='Mysql backup every hour',
schedule_interval='00 * * * *', start_date=datetime(2020,
11, 2), catchup=False)

operator = BashOperator(task_id='backup_task',
bash_command='mysqldump -u root -ppassword sample >
"/home/shrini/backups/"sample_"$(date +"%Y%m%d_%H%M
%S").sql"',dag=dag)
```

இதில் ஷெல் கட்டளைகள் நேரடியாக இயக்கப்படுவதைக் காணலாம்.
பொதுவாக dag-ன் பெயர்தான் திரையில் வெளிப்படும். ஆகவே dag-ன் பெயரையே புரோகிராமின் பெயராக வைத்துக்கொள்வது நல்லது.

புதிதாக bash ஆப்பரேட்டர் கொண்டு உருவாக்கப்பட்ட dag-ஐ திரையில் ரன் செய்து பார்க்கவும். அது தோல்வியுற்று சிகப்பு நிற வளையத்தை வெளிப்படுத்துவதைக் காணலாம்.

		DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
	On	bash_exp	00 * * * *	airflow	1	2020-11-09 06:28	2	

என்ன காரணம் என்று தெரிந்து கொள்ள அவ்வளையத்தின் மீது சொடுக்கினால் அதற்கான Run Id பின்வருமாறு வெளிப்படும்.

Dag Runs

List (2) Create Add Filter With selected Search: dag_id, state, run_id

× Dag Id equals bash_exp

× State equals failed

Reset

		State	Dag Id	Execution Date	Run Id	External Trigger
		failed	bash_exp	11-09T06:28:22.337168+00:00	manual__2020-11-09T06:28:22.337168+00:00	
		failed	bash_exp	11-09T05:00:00+00:00	scheduled__2020-11-09T05:00:00+00:00	

அந்த Id-ன் மீது சொடுக்கினால், அது பின்வருமாறு திரையை வெளிப்படுத்தும்.

அதில் view log-ன் மீது சொடுக்கவும். என்ன காரணம் என்பது பின்வருமாறு வெளிப்படுத்தப்பட்டுள்ளது.

```
[2020-11-09 11:59:08,149] {bash_operator.py:157} INFO -
mysqldump: Got error: 2002: Can't connect to local MySQL server
through socket '/var/run/mysqld/mysqld.sock' (2) when trying to
connect
```

அதாவது Mysql-ஐத் தொடர்பு கொள்வதில் ஏதோ சிக்கல் என்று வெளிப்படுத்தியுள்ளது. எனவே Mysql இயங்குகின்ற இடத்தில் சென்று அதற்கான சேவையைத் துவக்கி அதன் இயக்கத்தை உறுதிபடுத்திக் கொள்ளவும்.

இதுவரை பல்வேறு சர்வர்களில் ஓடிக்கொண்டிருக்கும் பணிகளை மேலாண்மை செய்ய உதவும் Airflow கருவியை பற்றிப் பார்த்தோம். அடுத்ததாக பல்வேறு சர்வர்களில் ஒரே நேரத்தில் பல

மென்பொருட்களை நிறுவுதல், மேம்படுத்துதல், நீக்குதல் போன்ற பல பணிகளைச் செய்ய உதவும் Ansible - ஐப் பற்றிப் பார்க்கலாம்.

12. Ansible

உங்களிடம் ஒரு லினக்ஸ் சர்வர் உள்ளது. இதில் இன்று நீங்கள் 8 மென்பொருட்களை நிறுவ வேண்டும். 15 மென்பொருட்களை மேம்படுத்த வேண்டும். 2 மென்பொருட்களை நீக்க வேண்டும். எப்படிச் செய்வீர்கள்? அவற்றுக்கான கட்டளைகளை ஒன்றன் பின் ஒன்றாகத்தான் தருவீர்கள். சரிதானே. இதே வேலையை 5 சர்வர்களில் செய்ய வேண்டும் என்றால்? ஒவ்வொரு சர்வராக login செய்து எல்லாக் கட்டளைகளையும் இயக்க வேண்டியதுதான். இதுவே 50 சர்வர், 100 சர்வர் என்றால்? ஒவ்வொன்றிலும் login செய்து அதே கட்டளைகளைத் திரும்பத்திரும்ப இயக்குவது என்பது சற்று கடினமான வேலை தான்.

Ansible என்பது ஒரே கணிணியில் இருந்துகொண்டே பல்வேறு கணினிகளில் உள்நுழைந்து பல்வேறு மென்பொருட்களை நிறுவுதல், நீக்குதல், மேம்படுத்துதல், நிர்வகித்தல் போன்ற பல கட்டளைகளை இயக்க உதவுகிறது. Bash script-ம் இதே வேலையைச் செய்தாலும் அதனிடம் சூழலை ஆராயும் திறன் இல்லை. உதாரணமாக ஒரு பெரிய bash script-ன் ஒரு பகுதியாக 10 மென்பொருட்களை நீக்கும் கட்டளை உள்ளது என வைத்துக் கொள்வோம். முதன்முறை அவற்றை இயக்கும்போது அதன் வேலையைச் செய்யும். அதே script-ஐ இரண்டாவது முறை இயக்கும்போது பிழைச்செய்தி வெளிப்பட்டு பாதியிலேயே நின்று விடும். ஏனெனில் அதைத்தான் ஏற்கெனவே நீக்கி விட்டதே. ஆகவே ஏற்கெனவே நீக்கப் பட்டிருந்தால், அவற்றை மீண்டும் நீக்க முயலக்கூடாது என்பதையும் நாம்தான் கட்டளையாக எழுத வேண்டும். இங்குதான் Ansible-ன் சூழலை ஆராயும் திறன் நமக்குப் பயன்படும். இதே வேலையை Ansible கொண்டு செய்யும் போது 10 மென்பொருட்களும் இருக்கின்றனவா, இல்லையா என்பதை ஆராய்ந்து, உண்மைகளைப் (Facts) பெற்று, அதனடிப்படையில் தனது வேலையைத் தொடங்கும். ஆகவே ஏற்கெனவே நீக்கப்பட்டிருந்தால் விட்டுவிடும். இல்லையெனில் நீக்கிவிடும். அவ்வளவுதான். இதனால், Ansible Script-களை எத்தனை முறை வேண்டுமானாலும் இயக்கிப் பார்க்கலாம். எந்தப் பிழையும் வெளிப்படுத்தாது.

மேலும் bash script-ல் ஒவ்வொரு சர்வருக்கும் ஏற்ப கட்டளைகள் மாறுபடும். சில சர்வர்களில் apt-get install, சிலவற்றில் yum install என ஒரே விஷயத்திற்குப் பல மாதிரி கட்டளைகள் இருக்கும். இப்பிரச்சினையையும் ansible லாவகமாகக் கையாள்கிறது. அனைத்து சர்வர்களுக்கும் பொதுவான வகையில் Module என்ற ஒன்றைப் பயன்படுத்தி அதன் மூலம் தனது வேலைகளைச் செய்கிறது.

இதுபோன்று பல கணினிகளில் திரும்பத்திரும்ப செய்யப்படும் வேலைகளை தானியக்கமாகச் செய்து முடிப்பதற்கு configuration management என்ற வகையிலான மென்பொருட்கள் பயன்படுகின்றன. அவற்றில் Puppet, chef, salt, ansible போன்றவை பிரபலமான கட்டற்ற மென்பொருட்கள் ஆகும். puppet, Chef, salt போன்றவை client-server முறையில் இயங்குபவை. ஒவ்வொரு கணிணியிலும் ஒரு client மென்பொருள் இயங்க வேண்டும். ஆனால் ansible-க்கு வெறும் ssh அனுமதி மட்டுமே போதும். அதன் வழியே login செய்து எல்லா வேலைகளையும் செய்துவிடும்.

Ansible கற்றுக் கொள்ள, நம்மிடம் பல சர்வர்கள் இருக்கவேண்டிய அவசியம் இல்லை. நம்மிடம் கணிணியே போதும். அவற்றிலேயே LXC Container மூலம் பல குட்டி மாயக்கணிணிகள் உருவாக்க முடியும். மேலும் VirtualBox, Vagrant, Xen மூலமும் பல மாயக் கணிணிகள் உருவாக்கி, அவற்றை Ansbile மூலம் பராமரிக்கலாம்.

12.1 Installation

Ansible முற்றிலும் பைதான் மொழியில் எழுதப்பட்டது. எல்லா GNU/Linux கணிணிகளிலும் பைதான் இயல்பாகவே இருக்கும்.

```
$ sudo pip3 install ansible
```

என்ற கட்டளை வழியே எளிதில் நிறுவலாம். நிறுவியவுடன் /etc/ansible/hosts என்ற இடத்தில் சென்று நாம் தொடர்பு கொள்ள

வேண்டிய அனைத்துக் கணினிகளின் ip முகவரிகளையும் குறிப்பிட வேண்டும்.

```
$ sudo vim /etc/ansible/hosts

[db]
139.59.47.5
35.166.185.40
```

```
#10.25.1.56
#10.25.1.57

# Here's another example of host rang
# leading 0s:

#db-[99:101]-node.example.com

[db]
139.59.47.5
35.166.185.40
"/etc/ansible/hosts" 47L, 1013C
```

இதற்கு db என்று ஒரு பொதுப்பெயர் இங்கு தரப்பட்டுள்ளது. இப்போதைக்கு இந்த பொதுப்பெயர் மட்டும் போதும். வேண்டுமெனில் பல groups, range of address, variable, dynamic inventory, fancy setup போன்ற பலவற்றையும் தரலாம். இதுவே Inventory file என்று அழைக்கப்படுகிறது.

Ansible பொதுவாக /etc/ansible/hosts என்ற கோப்பையே படிக்கிறது. வேறு இடத்தில் இந்தக் கோப்பை வைத்தால், அதன் இடத்தை தனியே குறிப்பிட்டுச் சொல்ல வேண்டும்.

இது Agentless என்ற முறையில் இயங்குவதால், இங்கு குறிப்பிட்டுள்ள முகவரிகளில் சென்று வேறு எதுவும் நிறுவத் தேவை இல்லை. அவற்றை ssh வழியே login செய்யும் அனுமதி மட்டும் இருந்தால் போதும்.

பொதுவாக Secured Shell (ssh) -ஐப் பயன்படுத்தி மற்ற கணினிகளுக்குள் நுழையும் போது ஒவ்வொரு முறையும் சரியான கடவுச்சொல்லை அளித்தால் மட்டுமே உள்நுழைய முடியும். Ansible மூலம் தானியக்க முறையில் உள்நுழையும் போது இவ்வாறு பாஸ்வேர்ட் கேட்டு நிற்கக்கூடாது. ஆகவே நமது கணினியின் ssh-க்கு பொதுத் திறவுகோல் என்று ஒன்று இருக்கும். அதற்கான சான்றிதழை எடுத்து நமது கணினி மற்றும் மற்ற கணினிகளின் அங்கீகரிக்கப்பட்ட திறவுகோல்களின் பட்டியலில் இணைக்க வேண்டும். இது பின்வருமாறு.

```
$ ssh-keygen
```

இக்கட்டளை நமது home directory-ல் .rsa எனும் folder-ஐ உருவாக்கி, அதற்குள் பின்வருமாறு கோப்புகளை உருவாக்கும்.

• id_rsa (தனிப்பட்ட திறவுகோல்)
• id_rsa.pub (பொதுவான திறவுகோல்)
• authorised_keys (அங்கீகரிக்கப்பட்ட திறவுகோல்கள்)

இதில் id_rsa.pub என்பதுதான் SSH-க்கான சான்றிதழ். இதனை நம் கணினியின் authorised_keys என்பதற்குள் போட வேண்டும்..

```
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

```
shrini@nithya-Lenovo-ideapad:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/shrini/.ssh/id_rsa):
/home/shrini/.ssh/id_rsa already exists.
Overwrite (y/n)? n
shrini@nithya-Lenovo-ideapad:~$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

அவ்வாறே மற்ற கணினிகளுக்குள்ளும் ssh மூலம் உள்நுழைந்து private key, public key, authorised_keys ஆகிய மூன்றையும் உருவாக்க வேண்டும். இங்கு பாஸ்வேர்ட் கொடுத்து உள்நுழைந்துள்ளோம்..

```
$ ssh shrini@139.59.47.5
$ ssh-keygen
```

```
shrini@punjabiwiki:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/shrini/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/shrini/.ssh/id_rsa.
Your public key has been saved in /home/shrini/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:aX/2XgnaGrGHrm/XGSVahrPJm+QKc31KgNlbM57IGfQ shrini@punjabiwiki
The key's randomart image is:
+---[RSA 2048]----+
|                 |
|                 |
|            ..   |
|         + oo.+ .|
|        S oooOE..|
|       . . +%B.=.|
|        o oBX==.+|
|         +.=B= + |
|          .==o.o |
+----[SHA256]-----+
```

இக்கணினியில் பாஸ்வேர்டுக்கு நிகரான pem file மூலம் உள்நுழைந்துள்ளோம்.

```
$ ssh -i /home/shrini/shrini-freepem.pem
ubuntu@35.166.185.40
$ ssh-keygen
```

```
(base) ubuntu@ip-172-31-29-250:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ubuntu/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ubuntu/.ssh/id_rsa.
Your public key has been saved in /home/ubuntu/.ssh/id_rsa.pub.
The key fingerprint is:
fd:d5:75:da:58:93:31:08:1b:7f:0c:9c:c6:e4:5d:16 ubuntu@ip-172-31-29-250
The key's randomart image is:
+--[ RSA 2048]----+
|           o=ooE+|
|            =*+.=|
|           ..o *+|
|        .     .*+|
|        S .   + o|
|           . .   |
|            .    |
|                 |
|                 |
+-----------------+
(base) ubuntu@ip-172-31-29-250:~$ ^C
(base) ubuntu@ip-172-31-29-250:~$ exit
logout
Connection to 35.166.185.40 closed.
```

பின் SCP மூலம் நம் கணினியின் பொதுத் திறவுகோலை இவ்விரு கணினிகளின் authorised_keys உள்ளும் போட வேண்டும்..

```
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$ scp ~/.ssh/id_rsa.pub
shrini@139.59.47.5:/home/shrini/.ssh/authorized_keys
$ scp -i /home/shrini/shrini-freepem.pem ~/.ssh/id_rsa.pub
ubuntu@35.166.185.40:/home/ubuntu/.ssh/authorized_keys
```

அவ்வளவுதான்! இனி மேல் ஒரே கணினியில் இருந்து கொண்டே மற்ற இரு கணினிகளுக்கு உள்ளும் கடவுச்சொல் இல்லாமலேயே உள்நுழைந்து வேண்டிய வேலைகளைச் செய்து கொள்ளலாம்..

12.2 Commands

இப்போது குறிப்பிட்ட சில கட்டளைகளை அனைத்து சர்வர்களிலும் இயக்கிப் பார்க்கலாமா?

Ansible-ஐப் பொறுத்த வரை ஒவ்வொரு பணிக்கும் ஒவ்வொரு Module-ஐப் பயன்படுத்துகிறது. Module மூலம் மென்பொருள் நிறுவுதல், கோப்புகளை நகல் எடுத்தல், உருவாக்குதல், திருத்துதல் என Commandline-ல் நாம் செய்யும் எதையும் செய்யலாம். நேரடியாக command மூலம் செய்யாமல், அவற்றுக்கான module மூலம் செய்தால், பலன்கள் அதிகம்.

இந்த Modules-தான் என்ன செய்ய வேண்டுமோ அவை ஏற்கனவே அக்கணினிகளில் செய்யப்பட்டு விட்டதா என்பதை ஆராய்ந்து உண்மைகளை(Facts) பெற்று, அதனடிப்படையில் செயல்களைத் தொடங்குவதற்கு ஆதாரமாக இருக்கின்றன.

ping என்ற கட்டளையை எல்லா சர்வர்களிலும் இயக்குவதற்கான ansible கட்டளை கீழ்க்கண்டவாறு அமையும். -m என்பதைத் தொடர்ந்து ping என்ற module-ஐக் கொடுப்பதன் மூலம் இதனை செய்கிறது. all என்பது அனைத்து சர்வர்களிலும் இந்த மாடியூலை இயக்க வேண்டும் என்பதைக் குறிக்கிறது.

```
$ ansible all -m ping
```

```
shrini@nithya-Lenovo-ideapad:~$ ansible all -m ping
35.166.185.40 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: shrini@35.166.185.40: Permission denied (publickey).",
    "unreachable": true
}
[DEPRECATION WARNING]: Distribution Ubuntu 18.04 on host 139.59.47.5 should use /usr/bin/python3, but is
Ansible releases. A future Ansible release will default to using the discovered platform python for this
https://docs.ansible.com/ansible/2.9/reference_appendices/interpreter_discovery.html for more informatio
warnings can be disabled by setting deprecation_warnings=False in ansible.cfg.
139.59.47.5 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

JSON வடிவில் இருக்கும் இதன் வெளியீட்டில் ஒரு சர்வரில் மட்டும் சென்று வெற்றிகரமாக ping செய்துள்ளதைக் காணலாம். 35.166.185.40 என்ற சர்வரைத் தொடர்பு கொள்ள முடியாமல் பிழைச்செய்தியை வெளிப்படுத்தியுள்ளது. ஏனெனில் shrini என்ற பயனராக Ansbile script-ஐ இயக்கும்போது, உங்கள் சர்வர்களிலும் அதே பயனராகவே login செய்யப்படும். எனவேதான் shrini என்ற பயனரைக் கொண்ட ஒரு கணினியில் கட்டளை வெற்றிகரமாக இயக்கப்பட்டுவிட்டது. ஆனால் மற்றொரு கணினியிலோ ubuntu என்ற பயனராக உள்நுழைய வேண்டும். அதனால்தான் பிழைச்செய்தி ஏற்பட்டுள்ளது. ஆகவே host file-ல் ஐ.பி முகவரியைக் குறிப்பிடும் இடத்திலேயே எந்தப் பயனராக உள்நுழைய வேண்டும் என்பதையும் குறிப்பிடுவதன் மூலம் இப்பிரச்சினையைத் தவிர்க்கலாம்..

```
$ vim /etc/ansible/hosts
35.166.185.40 ansible_user=ubuntu
```

```
[db]
139.59.47.5
35.166.185.40 ansible_user=ubuntu
```

மீண்டும் ping செய்து பார்க்கவும். இப்போது இரண்டு சர்வர்களிலும் கட்டளை வெற்றிகரமாக இயக்கப்பட்டு விட்டது..

```
shrini@nithya-Lenovo-ideapad:~$ ansible all -m ping
[DEPRECATION WARNING]: Distribution Ubuntu 18.04 on host 139.59.47.5
Ansible releases. A future Ansible release will default to using the
https://docs.ansible.com/ansible/2.9/reference_appendices/interprete
warnings can be disabled by setting deprecation_warnings=False in ans
139.59.47.5 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
35.166.185.40 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

அடுத்ததாக அனைத்து சர்வர்களிலும் சென்று date என்ற ஷெல் கட்டளையை இயக்குவதற்கான ansible கட்டளை கீழ்க்கண்டவாறு அமையும். இதில் -m என்பதைத் தொடர்ந்து ஷெல் மாடியூல் கொடுக்கப்பட்டுள்ளது. பின் அதன் argument-ஆக date என்ற கட்டளை அளிக்கப்பட்டுள்ளது.

```
$ ansible all -m shell -a 'date'
```

```
shrini@nithya-Lenovo-ideapad:~$ ansible all -m shell -a 'date'
[DEPRECATION WARNING]: Distribution Ubuntu 18.04 on host 139.59.47.5 shou
Ansible releases. A future Ansible release will default to using the disc
https://docs.ansible.com/ansible/2.9/reference_appendices/interpreter_dis
warnings can be disabled by setting deprecation_warnings=False in ansible
139.59.47.5 | CHANGED | rc=0 >>
Tue Nov 10 05:48:33 UTC 2020
35.166.185.40 | CHANGED | rc=0 >>
Tue Nov 10 05:51:58 UTC 2020
```

இது அனைத்து சர்வர்களிலும் சென்று date-ஐ வெளிப்படுத்தியுள்ளதைக் காணலாம்.

12.3 Playbook

கட்டளைகளை தனித்தனியே இயக்குவதற்கு பதிலாக ஒரு கோப்பில் எழுதி, அக்கோப்பினை இயக்குவதன் மூலம் அனைத்து வேலைகளையும் செய்து கொள்ளலாம். இதற்கு Playbook என்று பெயர். இது yaml வடிவில் .yml என்ற extension கொண்டு சேமிக்கப்படுகிறது. இதில் குறிப்பிடப்படுகின்ற ஒவ்வொரு வேலையும் task என்று அழைக்கப்படுகின்றன.
Ansible ஆனது இந்த Playbook-ல் உள்ளவற்றைப் படித்து, hosts-ல் உள்ள ஒவ்வொரு கணிணியாக login செய்து, தரப்பட்ட கட்டளைகளை இயக்கிவிடும். பின் நமது கணிணிக்குத் திரும்பிவிடும்.

https://gist.github.com/nithyadurai87/3d70b0aa288c2f398d4467f1053620d3

```
$ vim playbook1.yml

- hosts: all
  become: true
  tasks:
    - name: Update apt
      apt: update_cache=yes

    - name: Install mysql
      apt: name=mysql-client state=latest
```

```
- hosts: all
  become: true
  tasks:
    - name: Update apt
      apt: update_cache=yes

    - name: Install docker
      apt: name=docker state=latest


~
"playbook1.yml" 10L, 154C
```

நிரலுக்கான விளக்கம்:
hosts:all என்பது hosts கோப்பில் உள்ள எல்லாக் கணினிகளையும் குறிக்கும். இவ்வாறு அல்லாமல் ஒரு குறிப்பிட்ட பெயரை அளித்து அக்குழுவில் உள்ள கணினிகளில் மட்டும் இயக்குமாறும் செய்யலாம்.
become:true என்பது sudo-ஆக கட்டளைகளை இயக்க உதவும்.
tasks: இதன் கீழ் அனைத்து வேலைகளும் தொடர்ச்சியாக கொடுக்கப்படுகின்றன. ஒவ்வொன்றுக்கும் -name ஐத் தொடர்ந்து ஒரு பெயர் அளிக்கப்படுகிறது. பின் apt என்ற மாடியூலைப் பயன்படுத்தி apt-get update-ஐ இயக்கியபின் docker-ஐ நிறுவுவதற்கான கட்டளை கொடுக்கப்பட்டுள்ளது.

ansible-playbook என்ற கட்டளை மூலம் இதை நாம் இயக்கலாம்.

```
$ ansible-playbook playbook1.yml
```

```
shrini@nithya-Lenovo-ideapad:~$ ansible-playbook playbook1.yml

PLAY [all] *************************************************************************************

TASK [Gathering Facts] *************************************************************************
fatal: [139.59.47.5]: FAILED! => {"msg": "Missing sudo password"}
ok: [35.166.185.40]

TASK [Update apt] ******************************************************************************
changed: [35.166.185.40]

TASK [Install docker] **************************************************************************
ok: [35.166.185.40]

PLAY RECAP *************************************************************************************
139.59.47.5                : ok=0    changed=0    unreachable=0    failed=1    skipped=0    rescued=0    ignored=0
35.166.185.40              : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

வெளியீட்டின் விளக்கம்:

Ansible சூழலை ஆராயும் என்று ஏற்கெனவே பார்த்தோம் அல்லவா! அதேபோல் முதல் வேலையாக சூழலை ஆராய்ந்து TASK [Gathering Facts] என்பதன் கீழ், ஒரு சர்வரில் எல்லாம் சரியாக உள்ளது. மற்றொரு சர்வரில் sudo கொண்டு கட்டளையை இயக்கும் போது பாஸ்வேர்ட் கேட்டு நிற்கிறது என்பதை வெளிப்படுத்தியுள்ளது. இதுவே failed=1 என கடைசியில் வெளிப்படக் காரணம். ஆகவே எந்த சர்வரில் எல்லாம் சரியாக உள்ளதோ அதில் மட்டும் சென்று மற்ற இரண்டு வேலைகளை செய்கிறது. பொதுவாக Ansible செய்ய வேண்டிய வேலை அக்கணினியில் ஏற்கனவே செய்யப்பட்டிருந்தால் அதனை ok எனவும், அவ்வாறில்லாமல் Ansible மூலம் செய்யப்பட்டால் அதனை changed எனவும் வெளிப்படுத்தும். இங்கு Update apt எனும் வேலையின் கீழ் changed எனவும், Install docker எனும் வேலையின் கீழ் ok எனவும் வெளிப்படுத்தியுள்ளதைக் காணலாம். அதாவது update apt கட்டளையை இயக்கி வேலை செய்துள்ளதால் அதனை changed எனவும், docker ஏற்கனவே இன்ஸ்டால் செய்யப்பட்டிருப்பதால் அதனை ok எனவும் வெளிப்படுத்தியுள்ளது.

அடுத்து failed என வந்துள்ள சர்வரில் சென்று sudo கொண்டு கட்டளையை இயக்கிப் பார்க்கவும்.

```
$ ssh shrini@139.59.47.5
$ sudo apt-get install docker
```

```
Last login: Tue Nov 10 10:38:54 2020 from 171.76.
shrini@punjabiwiki:~$ sudo apt-get install docker
[sudo] password for shrini:
```

ஆம். பாஸ்வேர்ட் கேட்டு நிற்கிறது. எனவே தான் Ansible-ஆல் இந்த சர்வரில் எதுவும் செய்ய முடியவில்லை. இதைத் தவிர்க்க /etc/sudoers.d எனும் கோப்பில் சில மாற்றங்களை செய்ய வேண்டும். இது பின்வருமாறு.

```
$ sudo visudo
shrini ALL=(ALL) NOPASSWD:ALL
```

```
%admin ALL=(ALL) ALL

# Allow members of group sudo
%sudo   ALL=(ALL:ALL) ALL
shrini ALL=(ALL) NOPASSWD:ALL
# See sudoers(5) for more info

#includedir /etc/sudoers.d
```

இப்போது sudo கட்டளையை இயக்கிப் பார்த்தால், அது பாஸ்வேர்ட் எதுவும் கேட்காமல் இயங்குவதைக் காணலாம்.

```
shrini@punjabiwiki:~$ sudo visudo
shrini@punjabiwiki:~$ sudo apt-get install docker
E: dpkg was interrupted, you must manually run 'sudo dpkg --configure -a' to correct the problem.
shrini@punjabiwiki:~$ sudo dpkg --configure -a
Setting up linux-headers-4.15.0-108 (4.15.0-108.109) ...
shrini@punjabiwiki:~$ sudo apt-get install docker
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  linux-headers-4.15.0-108 linux-headers-4.15.0-108-generic
```

ஆகவே மீண்டும் ஒருமுறை playbook-ஐ இயக்கவும். அது எவ்விதத் தடையும் இன்றி இரு சர்வர்களிலும் கட்டளைகளை இயக்கி முடிப்பதைக் காணலாம்..

12.4 Modules

அடுத்ததாக பல்வேறு வேலைகளைச் செய்ய உதவும் பல்வேறு Modules-ஐப் பற்றி இந்தப் பகுதியில் காணலாம். அனைத்து சர்வர்களிலும் சென்று ஒரு டைரக்டரியை உருவாக்குதல், நீக்குதல் போன்ற விஷயங்களை மாடியூல் கொண்டு எவ்வாறு செய்வது என playbook2.yml எனும் கோப்பிற்குள் கொடுத்துள்ளோம். மேலும் kafka-வை பதிவிறக்கம் செய்து extract செய்தல், அதன் config கோப்புகளில் மாற்றம் செய்தல் போன்ற விஷயங்களை செய்வதற்கான

கட்டளைகளை ஏற்கனவே பார்த்தோம். அவற்றை மாடியூல் கொண்டு எவ்வாறு செய்வது எனவும் இங்கு குறிப்பிட்டுள்ளோம். இவை பின்வருமாறு.

https://gist.github.com/nithyadurai87/1bb70558e7d0abab912aaf945b4c03d8

```
$ vim playbook2.yml

- hosts: all
  become: true
  tasks:

    - name: create directory
      file: state=directory path=kafka_test
      file: state=directory path=/opt/kafka_test

    - name: delete directory
      file: state=absent path=kafka_test

    - name: Download kafka
      get_url:
        url: https://downloads.apache.org/kafka/2.5.0/kafka_2.12-2.5.0.tgz
        dest: /opt/kafka_test/

    - name: Extract archive
      unarchive:
        src: /opt/kafka_test/kafka_2.12-2.5.0.tgz
        dest: /opt/kafka_test
        remote_src: yes

    - name: Edit config file
      lineinfile:
        path: /opt/kafka_test/kafka_2.12-2.5.0/config/server.properties
        line: listeners=PLAINTEXT://localhost:9092
        insertbefore: BOF
```

```
- hosts: all
  become: true
  tasks:

    - name: create directory
      file: state=directory path=kafka_test
      file: state=directory path=/opt/kafka_test

    - name: delete directory
      file: state=absent path=kafka_test

    - name: Download kafka
      get_url:
        url: https://downloads.apache.org/kafka/2.5.0/kafka_2.12-2.5.0.tgz
        dest: /opt/kafka_test/

    - name: Extract archive
      unarchive:
        src: /opt/kafka_test/kafka_2.12-2.5.0.tgz
        dest: /opt/kafka_test
        remote_src: yes

    - name: Edit config file
      lineinfile:
        path: /opt/kafka_test/kafka_2.12-2.5.0/config/server.properties
        line: listeners=PLAINTEXT://localhost:9092
        insertbefore: BOF
```

நிரலுக்கான விளக்கம்:

இதில் kafka_test எனும் டைரக்டரி, ஹோம் ஃபோல்டரின் கீழும், /opt ஃபோல்டரின் கீழும் உருவாக்கப்பட்டுள்ளது. பின் ஹோம் ஃபோல்டரின் கீழ் உருவானது நீக்கப்படுகிறது. இவ்விரண்டு செயல்களுக்கும் file எனும் மாடியூல் பயன்படுகிறது. பின் get_url எனும் மாடியூலுக்கு எந்த முகவரியிலிருந்து பதிவிறக்கம் செய்ய வேண்டும், எந்த இடத்தில் செய்ய வேண்டும் என்பதை parameters-ஆக அளிப்பதன் மூலம் kafka பதிவிறக்கம் செய்யப்படுகிறது. unarchive என்ற மாடியூல் மூலம் extract செய்யப்படுகிறது. பின் அதன் config ஃபோல்டருக்குள் server.properties எனும் கோப்பில் listeners-ன் மதிப்பை localhost என அமைப்பதற்கு lineinfile எனும் மாடியூல் பயன்படுகிறது.

ansible-playbook என்ற கட்டளை மூலம் இதனை இயக்கவும்.

```
$ ansible-playbook playbook2.yml
```

அதன் வெளியீடு பின்வருமாறு.

```
TASK [create directory] ******************************************************************************
ok: [139.59.47.5]
ok: [35.166.185.40]

TASK [delete directory] ******************************************************************************
changed: [139.59.47.5]
changed: [35.166.185.40]

TASK [Download kafka] ********************************************************************************
ok: [139.59.47.5]
ok: [35.166.185.40]

TASK [Extract archive] *******************************************************************************
changed: [139.59.47.5]
changed: [35.166.185.40]

TASK [Edit config file] ******************************************************************************
changed: [139.59.47.5]
changed: [35.166.185.40]

PLAY RECAP *******************************************************************************************
139.59.47.5                : ok=6    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
35.166.185.40              : ok=6    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

12.5 Variables & Handlers

Playbook2-ல் குறிப்பிட்டுள்ள அனைத்து வேலைகளுக்கும் வெவ்வேறு மாடியூல்களைப் பயன்படுத்தாமல், வெறும் ஷெல் மாடியூலை மட்டும் வைத்துக் கூட எழுதலாம். இது பின்வருமாறு.

https://gist.github.com/nithyadurai87/b5e4d94f36ca08f38dba8de470b22e43

```
$ vim playbook3.yml

- hosts: all
  vars:
    url: "https://downloads.apache.org/kafka/2.5.0/kafka_2.12-2.5.0.tgz"
  become: true
  tasks:
```

```
    - name: kafka steps
      shell:
        cmd: mkdir kafka_test
        cmd: mkdir /opt/kafka_test
        cmd: rmdir kafka_test
        cmd: cd /opt/kafka_test
        cmd: wget "{{url}}"
        cmd: tar -xzf kafka_2.12-2.5.0.tgz
        cmd: cd kafka_2.12-2.5.0/config/
        cmd: echo "listeners=PLAINTEXT://localhost:9092" >> server.properties
      notify:
        - start services
        - stop services

  handlers:
    - name: start services
      shell:
        cmd: /opt/kafka_test/kafka_2.12-2.5.0/bin/zookeeper-server-start.sh
/opt/kafka_test/kafka_2.12-2.5.0/config/zookeeper.properties &
    - name: stop services
      shell:
        cmd: exit
```

ஆனால், இந்த முறை சரியானது அல்ல. ஒரு bash script செய்யும் வேலையை மட்டுமே இது செய்கிறது. Indempotence அதாவது ஒரே வேலையை எந்த பாதிப்பும் இன்றி, எத்தனை முறை வேண்டுமானாலும் இயக்கிப் பார்க்கும் வசதியை module-கள் மட்டுமே தருகின்றன. சரியான module-ஐப் பயன்படுத்துவதன் மூலம், எந்தப் பிழையும் இன்றி கட்டளைகள் இயங்குவதை உறுதிபடுத்தலாம்.

இதில் variables மற்றும் handlers ஆகியவை பயன்படுத்தப்பட்டுள்ளது. vars எனும் பகுதியின் கீழ் variable-களை வரையறுக்கலாம். இங்கு url எனும் variable வரையறுக்கப்பட்டு, அது wget எனும் கட்டளையைத் தொடர்ந்து பயன்படுத்தப்பட்டுள்ளது.

Handlers என்பதும் Task போலத்தான். ஒரு வேலையைச் செய்ய உதவும். ஆனால், இதை வேறு ஒரு Task வழியாகவே இயக்கமுடியும். அதாவது ஒரு செயலுக்காகக் காத்திருந்து, அது நடந்தவுடன் தன் செயலைத் துவங்கும். Event System போல. இதன் மூலம் பல துணைச் செயல்களைச் செய்யலாம். உதாரணம், ஒரு Network service

நிறுவியவுடன், அதைத் தொடங்குதல். ஒரு configuration file மாற்றியவுடன், அதன் service ஐ reload செய்தல் போன்றவை. இங்கு நாம் எழுதிய handlers ன் பெயர் start services, stop services. இதை kafka- வின் config file- ஐ மாற்றியவுடன் நாம் அழைத்துள்ளோம்.. ஒரு Task ல் Notify பகுதி இருந்தால், Task முடிந்தவுடன், Notify ல் கூறப்பட்டுள்ள handler-ஆனது இயக்கப்படுகிறது. அது முடிந்தவுடன் மீண்டும் Task-ல் கூறப்பட்டுள்ள அடுத்த வேலையைச் செய்ய சென்று விடுகிறது. ஆகவே இதனை ஒரு function-க்குச் சமமாகக் கூறலாம்.

இதனை இயக்குவதற்கான கட்டளை பின்வருமாறு.

```
$ ansible-playbook playbook3.yml
```

அதன் வெளியீடு

```
shrini@nithya-Lenovo-ideapad:~$ ansible-playbook playbook3.yml
[WARNING]: While constructing a mapping from /home/shrini/playbook3.yml, line 9, column 9, found a duplicate dict k

PLAY [all] ******************************************************************************************************

TASK [Gathering Facts] ******************************************************************************************
[DEPRECATION WARNING]: Distribution Ubuntu 18.04 on host 139.59.47.5 should use /usr/bin/python3, but is using /usr
Ansible releases. A future Ansible release will default to using the discovered platform python for this host. See
https://docs.ansible.com/ansible/2.9/reference_appendices/interpreter_discovery.html for more information. This fea
warnings can be disabled by setting deprecation_warnings=False in ansible.cfg.
ok: [139.59.47.5]
ok: [35.166.185.40]

TASK [kafka steps] **********************************************************************************************
changed: [139.59.47.5]
changed: [35.166.185.40]

RUNNING HANDLER [start services] ********************************************************************************
changed: [139.59.47.5]
changed: [35.166.185.40]

RUNNING HANDLER [stop services] *********************************************************************************
changed: [139.59.47.5]
changed: [35.166.185.40]

PLAY RECAP ******************************************************************************************************
139.59.47.5                : ok=4    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
35.166.185.40              : ok=4    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

12.6 Roles

Modules, Variables, Handlers போன்ற அனைத்தையும் ஒரே playbook-இன் பல்வேறு பகுதிகளாக வரையறுக்காமல் Role என்ற ஒன்றை உருவாக்கி அதன்கீழ் அமையும் கோப்புகளுக்குள் ஒவ்வொன்றையும் தனித்தனியே வரையறுக்கலாம். பின் மற்றொரு playbook வழியே இந்த role-ஐ இயக்குவதன் மூலம் அனைத்தையும் ஒன்றாக இயக்கலாம்.

ansible-galaxy என்ற கட்டளை Role- ஐ உருவாக்கப் பயன்படுகிறது. இங்கு playbook4 எனும் பெயரில் role உருவாக்கப்பட்டுள்ளது. அதன்கீழ் பல்வேறு directories அமைந்திருப்பதைக் காணலாம்.

```
$ ansible-galaxy init playbook4
```

```
shrini@nithya-Lenovo-ideapad:~$ ansible-galaxy init playbook4
- Role playbook4 was created successfully
shrini@nithya-Lenovo-ideapad:~$ cd playbook4/
shrini@nithya-Lenovo-ideapad:~/playbook4$ ls
defaults  files  handlers  meta  README.md  tasks  templates  tests  vars
```

ஒவ்வொரு டைரக்டரியின் கீழும் பல்வேறு விஷயங்களுக்கான yml கோப்புகள் காணப்படும்.

```
$ tree playbook4
```

```
shrini@nithya-Lenovo-ideapad:~$ tree playbook4
playbook4
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml

8 directories, 8 files
```

மேற்கண்ட பகுதியில் நாம் மொத்தமாக ஒரே பிளேபுக்கில் எழுதிய நிரல்களை இப்போது தனித்தனியாகப் பிரித்து அதற்கான கோப்பில் சென்று தனித்தனியாக எழுதி சேமிக்கவும்.

```
shrini@nithya-Lenovo-ideapad:~$ cat playbook4/tasks/main.yml
---
# tasks file for playbook4
    - name: kafka steps
      shell:
        cmd: mkdir kafka_test
        cmd: mkdir /opt/kafka_test
        cmd: rmdir kafka_test
        cmd: cd /opt/kafka_test
        cmd: wget "{{url}}"
        cmd: tar -xzf kafka_2.12-2.5.0.tgz
        cmd: cd kafka_2.12-2.5.0/config/
        cmd: echo "listeners=PLAINTEXT://localhost:9092" >> server.properties
      notify:
        - start services
        - stop services
```

```
shrini@nithya-Lenovo-ideapad:~$ cat playbook4/vars/main.yml
---
# vars file for playbook4
url: "https://downloads.apache.org/kafka/2.5.0/kafka_2.12-2.5.0.tgz"
```

```
shrini@nithya-Lenovo-ideapad:~$ cat playbook4/handlers/main.yml
---
# handlers file for playbook4
    - name: start services
      shell:
        cmd: /opt/kafka_test/kafka_2.12-2.5.0/bin/zookeeper-server-sta
    - name: stop services
      shell:
        cmd: exit
```

கடைசியாக ஒரு புதிய playbook-ஐ உருவாக்கி அதற்குள் நம்முடைய role- ஐப் பின்வருமாறு அழைப்பதன் மூலம் அனைத்து செயல்களையும் செய்து விடலாம்.

```
shrini@nithya-Lenovo-ideapad:~$ cat playbook5.yml
- hosts: all
  become: true

  roles:
    - role: playbook4
```

இதன் வெளியீடு பின்வருமாறு.

```
shrini@nithya-Lenovo-ideapad:~$ ansible-playbook playbook5.yml
[WARNING]: While constructing a mapping from
/home/shrini/playbook4/tasks/main.yml, line 5, column 9, found a duplicate dict
key (cmd). Using last defined value only.

PLAY [all] *********************************************************************

TASK [Gathering Facts] *********************************************************
[DEPRECATION WARNING]: Distribution Ubuntu 18.04 on host 139.59.47.5 should use
 /usr/bin/python3, but is using /usr/bin/python for backward compatibility with
 prior Ansible releases. A future Ansible release will default to using the
discovered platform python for this host. See https://docs.ansible.com/ansible/
2.9/reference_appendices/interpreter_discovery.html for more information. This
feature will be removed in version 2.12. Deprecation warnings can be disabled
by setting deprecation_warnings=False in ansible.cfg.
ok: [139.59.47.5]
ok: [35.166.185.40]

TASK [playbook4 : kafka steps] *************************************************
changed: [139.59.47.5]
changed: [35.166.185.40]

RUNNING HANDLER [playbook4 : start services] ***********************************
changed: [139.59.47.5]
changed: [35.166.185.40]

RUNNING HANDLER [playbook4 : stop services] ************************************
changed: [139.59.47.5]
changed: [35.166.185.40]

PLAY RECAP *********************************************************************
139.59.47.5                : ok=4    changed=3    unreachable=0    failed=0    s
```

13. காணொளிகள்

பின் வரும் YouTube Playlist ல் எனது காணொளிகளைக் காணலாம்

14. ஆசிரியரின் பிற மின்னூல்கள்

https://freetamilebooks.com/authors/nithyaduraisamy/



15. கணியம் பற்றி

இலக்குகள்

கட்டற்ற கணிநுட்பத்தின் எளிய விஷயங்கள் தொடங்கி அதிநுட்பமான அம்சங்கள் வரை அறிந்திட விழையும் எவருக்கும் தேவையான தகவல்களை தொடர்ச்சியாகத் தரும் தளமாய் உருபெறுவது.

- உரை, ஒலி, ஒளி என பல்லூடக வகைகளிலும் விவரங்களை தருவது.
- இத்துறையின் நிகழ்வுகளை எடுத்துரைப்பது.
- எவரும் பங்களிக்க ஏதுவாய் யாவருக்குமான நெறியில் விவரங்களை வழங்குவது.
- அச்சு வடிவிலும், புத்தகங்களாகவும், வட்டுக்களாகவும் விவரங்களை வெளியிடுவது.

பங்களிக்க

- விருப்பமுள்ள எவரும் பங்களிக்கலாம்.
- கட்டற்ற கணிநுட்பம் சார்ந்த விஷயமாக இருத்தல் வேண்டும்.

- பங்களிக்கத் தொடங்கும் முன்னர் கணியத்திற்கு உங்களுடைய பதிப்புரிமத்தை அளிக்க எதிர்பார்க்கப்படுகிறீர்கள்.
- **editor@kaniyam.com** முகவரிக்கு கீழ்க்கண்ட விவரங்களடங்கிய மடலொன்றை உறுதிமொழியாய் அளித்துவிட்டு யாரும் பங்களிக்கத் தொடங்கலாம்.

  - **மடலின் பொருள்:** பதிப்புரிமம் அளிப்பு
  - **மடல் உள்ளடக்கம்**

    - என்னால் கணியத்திற்காக அனுப்பப்படும் படைப்புகள் அனைத்தும் கணியத்திற்காக முதன்முதலாய் படைக்கப்பட்டதாக உறுதியளிக்கிறேன்.
    - இதன்பொருட்டு எனக்கிருக்கக்கூடிய பதிப்புரிமத்தினை கணியத்திற்கு வழங்குகிறேன்.
    - உங்களுடயை முழுப்பெயர், தேதி.

- தாங்கள் பங்களிக்க விரும்பும் ஒரு பகுதியில் வேறொருவர் ஏற்கனவே பங்களித்து வருகிறார் எனின் அவருடன் இணைந்து பணியாற்ற முனையவும்.
- கட்டுரைகள் மொழிபெயர்ப்புகளாகவும், விஷயமறிந்த ஒருவர் சொல்லக் கேட்டு கற்று இயற்றப்பட்டவையாகவும் இருக்கலாம்.
- படைப்புகள் தொடர்களாகவும் இருக்கலாம்.

- தொழில் நுட்பம், கொள்கை விளக்கம், பிரச்சாரம், கதை, கேலிச்சித்திரம், நையாண்டி எனப் பலசுவைகளிலும் இத்துறைக்கு பொருந்தும்படியான ஆக்கங்களாக இருக்கலாம்.
- தங்களுக்கு இயல்பான எந்தவொரு நடையிலும் எழுதலாம்.
- தங்களது படைப்புகளை எளியதொரு உரை ஆவணமாக editor@kaniyam.com முகவரிக்குஅனுப்பிவைக்கவும்.
- தள பராமரிப்பு, ஆதரவளித்தல் உள்ளிட்ட ஏனைய விதங்களிலும் பங்களிக்கலாம்.
- ஐயங்களிருப்பின் editor@kaniyam.com மடலியற்றவும்.

விண்ணப்பங்கள்

- கணித் தொழில்நுட்பத்தை அறிய விழையும் மக்களுக்காக மேற்கொள்ளப்படும் முயற்சியாகும் இது.
- இதில் பங்களிக்க தாங்கள் அதிநுட்ப ஆற்றல் வாய்ந்தவராக இருக்க வேண்டும் என்ற கட்டாயமில்லை.
- தங்களுக்கு தெரிந்த விஷயத்தை இயன்ற எளிய முறையில் எடுத்துரைக்க ஆர்வம் இருந்தால் போதும்.
- இதன் வளர்ச்சி நம் ஒவ்வொருவரின் கையிலுமே உள்ளது.
- குறைகளிலிருப்பின் முறையாக தெரியப்படுத்தி முன்னேற்றத்திற்கு வழி வகுக்கவும்.

வெளியீட்டு விவரம்

ஆசிரியர்: த. சீனிவாசன் – editor@kaniyam.com +91 98417 95468

16. கணியம் அறக்கட்டளை



16.1 தொலை நோக்கு – Vision

தமிழ் மொழி மற்றும் இனக்குழுக்கள் சார்ந்த மெய்நிகர்வளங்கள், கருவிகள் மற்றும் அறிவுத்தொகுதிகள், அனைவருக்கும் கட்டற்ற அணுக்கத்தில் கிடைக்கும் சூழல்

16.2 பணி இலக்கு – Mission

அறிவியல் மற்றும் சமூகப் பொருளாதார வளர்ச்சிக்கு ஒப்ப, தமிழ் மொழியின் பயன்பாடு வளர்வதை உறுதிப்படுத்துவதும், அனைத்து அறிவுத் தொகுதிகளும், வளங்களும் கட்டற்ற அணுக்கத்தில் அனைவருக்கும் கிடைக்கச்செய்தலும்.

16.3 தற்போதைய செயல்கள்

- கணியம் மின்னிதழ் – kaniyam.com
- கிரியேட்டிவ் காமன்சு உரிமையில் இலவச தமிழ் மின்னூல்கள் – FreeTamilEbooks.com

16.4 கட்டற்ற மென்பொருட்கள்

- உரை ஒலி மாற்றி – Text to Speech
- எழுத்துணரி – Optical Character Recognition
- விக்கிமூலத்துக்கான எழுத்துணரி
- மின்னூல்கள் கிண்டில் கருவிக்கு அனுப்புதல் – Send2Kindle
- விக்கிப்பீடியாவிற்கான சிறு கருவிகள்
- மின்னூல்கள் உருவாக்கும் கருவி
- உரை ஒலி மாற்றி – இணைய செயலி
- சங்க இலக்கியம் – ஆன்டிராய்டு செயலி
- FreeTamilEbooks – ஆன்டிராய்டு செயலி
- FreeTamilEbooks – ஐஓஎஸ் செயலி

- WikisourceEbooksReport இந்திய மொழிகளுக்ககான விக்கிமூலம் மின்னூல்கள் பதிவிறக்கப் பட்டியல்
- FreeTamilEbooks.com – Download counter மின்னூல்கள் பதிவிறக்கப் பட்டியல்

16.5 அடுத்த திட்டங்கள்/மென்பொருட்கள்

- விக்கி மூலத்தில் உள்ள மின்னூல்களை பகுதிநேர/முழு நேரப் பணியாளர்கள் மூலம் விரைந்து பிழை திருத்துதல்
- முழு நேர நிரலரை பணியமர்த்தி பல்வேறு கட்டற்ற மென்பொருட்கள் உருவாக்குதல்
- தமிழ் NLP க்கான பயிற்சிப் பட்டறைகள் நடத்துதல்
- கணியம் வாசகர் வட்டம் உருவாக்குதல்
- கட்டற்ற மென்பொருட்கள், கிரியேட்டிவ் காமன்சு உரிமையில் வளங்களை உருவாக்குபவர்களைக் கண்டறிந்து ஊக்குவித்தல்
- கணியம் இதழில் அதிக பங்களிப்பாளர்களை உருவாக்குதல், பயிற்சி அளித்தல்
- மின்னூலாக்கத்துக்கு ஒரு இணையதள செயலி
- எழுத்துணரிக்கு ஒரு இணையதள செயலி
- தமிழ் ஒலியோடைகள் உருவாக்கி வெளியிடுதல்

- OpenStreetMap.org ல் உள்ள இடம், தெரு, ஊர் பெயர்களை தமிழாக்கம் செய்தல்

- தமிழ்நாடு முழுவதையும் OpenStreetMap.org ல் வரைதல்

- குழந்தைக் கதைகளை ஒலி வடிவில் வழங்குதல்

- Ta.wiktionary.org ஐ ஒழுங்குபடுத்தி API க்கு தோதாக மாற்றுதல்

- Ta.wiktionary.org க்காக ஒலிப்பதிவு செய்யும் செயலி உருவாக்குதல்

- தமிழ் எழுத்துப் பிழைத்திருத்தி உருவாக்குதல்

- தமிழ் வேர்ச்சொல் காணும் கருவி உருவாக்குதல்

- எல்லா FreeTamilEbooks.com மின்னூல்களையும் Google Play Books, GoodReads.com ல் ஏற்றுதல்

- தமிழ் தட்டச்சு கற்க இணைய செயலி உருவாக்குதல்
- தமிழ் எழுதவும் படிக்கவும் கற்ற இணைய செயலி உருவாக்குதல் (aamozish.com/Course_preface போல)

மேற்கண்ட திட்டங்கள், மென்பொருட்களை உருவாக்கி செயல்படுத்த உங்கள் அனைவரின் ஆதரவும் தேவை. உங்களால் எவ்வாறேனும் பங்களிக்க இயலும் எனில் உங்கள் விவரங்களை kaniyamfoundation@gmail.com க்கு மின்னஞ்சல் அனுப்புங்கள்.

16.6 வெளிப்படைத்தன்மை

கணியம் அறக்கட்டளையின் செயல்கள், திட்டங்கள், மென்பொருட்கள் யாவும் அனைவருக்கும் பொதுவானதாகவும், 100% வெளிப்படைத்தன்மையுடனும் இருக்கும். இந்த இணைப்பில் செயல்களையும், இந்த இணைப்பில் மாத அறிக்கை, வரவு செலவு விவரங்களுடனும் காணலாம்.

கணியம் அறக்கட்டளையில் உருவாக்கப்படும் மென்பொருட்கள் யாவும் கட்டற்ற மென்பொருட்களாக மூல நிரலுடன், GNU GPL, Apache, BSD, MIT, Mozilla ஆகிய உரிமைகளில் ஒன்றாக வெளியிடப்படும். உருவாக்கப்படும் பிற வளங்கள், புகைப்படங்கள், ஒலிக்கோப்புகள், காணொளிகள், மின்னூல்கள், கட்டுரைகள் யாவும் யாவரும் பகிரும், பயன்படுத்தும் வகையில் கிரியேட்டிவ் காமன்சு உரிமையில் இருக்கும்.

16.7 நன்கொடை

உங்கள் நன்கொடைகள் தமிழுக்கான கட்டற்ற வளங்களை உருவாக்கும் செயல்களை சிறந்த வகையில் விரைந்து செய்ய ஊக்குவிக்கும்.

பின்வரும் வங்கிக் கணக்கில் உங்கள் நன்கொடைகளை அனுப்பி, உடனே விவரங்களை kaniyamfoundation@gmail.com க்கு மின்னஞ்சல் அனுப்புங்கள்.

Kaniyam Foundation
Account Number : 606 1010 100 502 79

Union Bank Of India
West Tambaram, Chennai

IFSC – UBIN0560618

Account Type : Current Account

16.8 UPI செயலிகளுக்கான QR Code



குறிப்பு: சில UPI செயலிகளில் இந்த QR Code வேலை செய்யாமல் போகலாம். அச்சமயம் மேலே உள்ள வங்கிக் கணக்கு எண், IFSC code ஐ பயன்படுத்தவும்.

Note: Sometimes UPI does not work properly, in that case kindly use Account number and IFSC code for internet banking.